Laufzeitadaption von zustandsbehafteten Datenstromoperatoren

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Dipl.-Ing. Bernhard Wolf

Betreuender Hochschullehrer:	Prof. Dr.-Ing. habil. Klaus Kabitzsch Technische Universität Dresden Fakultät Informatik, Institut für Angewandte Informatik Professur für Technische Informationssysteme
Gutachter:	Prof. Dr.-Ing. habil. Klaus Meyer-Wegener Universität Erlangen-Nürnberg Technische Fakultät, Department Informatik Lehrstuhl für Informatik 6 (Datenmanagement)
Tag der Verteidigung:	10. Dezember 2012

Dresden, im Dezember 2013

Kurzfassung

Änderungen von Datenstromanfragen zur Laufzeit werden insbesondere durch zustandsbehaftete Datenstromoperatoren erschwert. Da die Zustände im Arbeitsspeicher abgelegt sind und bei einem Neustart verloren gehen, wurden in der Vergangenheit Migrationsverfahren entwickelt, um die inneren Operatorzustände bei einem Änderungsvorgang zu erhalten. Die Migrationsverfahren basieren auf zwei unterschiedlichen Ansätzen – Zustandstransfer und Parallelausführung – sind jedoch aufgrund ihrer Realisierung auf eine zentrale Ausführung beschränkt.

Mit wachsenden Anforderungen in Bezug auf Datenmengen und Antwortzeiten werden Datenstromsysteme vermehrt verteilt ausgeführt, beispielsweise durch Sensornetze oder verteilte IT-Systeme. Zur Anpassung der Anfragen zur Laufzeit sind existierende Migrationsstrategien nicht oder nur bedingt geeignet. Diese Arbeit leistet einen Beitrag zur Lösung dieser Problematik und zur Optimierung der Migration in Datenstromsystemen.

Am Beispiel von präventiven Instandhaltungsstrategien in Fabrikumgebungen werden Anforderungen für die Datenstromverarbeitung und insbesondere für die Migration abgeleitet. Das generelle Ziel ist demnach eine möglichst schnelle Migration bei gleichzeitiger Ergebnisausgabe. In einer detaillierten Analyse der existierenden Migrationsstrategien werden deren Stärken und Schwächen bezüglich der gestellten Anforderungen diskutiert.

Für die Adaption von laufenden Datenstromanfragen wird eine allgemeine Methodik vorgestellt, welche als Basis für die neuen Strategien dient. Diese Adaptionsmethodik unterstützt zwei Verfahren zur Bestimmung von Migrationskonfigurationen – ein numerisches Verfahren für periodische Datenströme und ein heuristisches Verfahren, welches auch auf aperiodische Datenströme angewendet werden kann. Eine wesentliche Funktionalität zur Minimierung der Migrationsdauer ist dabei die Beschränkung auf notwendige Zustandswerte, da in verteilten Umgebungen eine Übertragungszeit für den Zustandstransfer veranschlagt werden muss – zwei Aspekte, die bei existierenden Verfahren nicht berücksichtigt werden. Durch die Verwendung von neu entwickelten Zustandstransfermethoden kann zudem die Übertragungsreihenfolge der einzelnen Zustandswerte beeinflusst werden.

Die Konzepte wurden in einem OSGi-basierten Prototyp implementiert und zudem simulativ analysiert. Mit einer umfassenden Evaluierung wird die Funktionsfähigkeit aller Komponenten und Konzepte demonstriert. Der Performance-Vergleich zwischen den existierenden und den neuen Migrationsstrategien fällt deutlich zu Gunsten der neuen Strategien aus, die zudem in der Lage sind, alle Anforderungen zu erfüllen.

Inhaltsverzeichnis

Abkürzungen und Formelzeichen

Abkürzungen

ACDS	Adapting Computational Data Streams [IS00a, IS00b], ein Datenstromsystem
CAPE	Constraint-exploiting Adaptive Processing Engine [CAP09, RDZ+05], ein Datenstromsystem
CF	Crest-Faktor, Scheitelfaktor
D-CAPE	Distributed Continuous Adaptive Processing System [CAP09, SR04], ein verteiltes Datenstromsystem
DB	Datenbank
DSID	Datenstrom-Identifikationsnummer
DSMS	Data Stream Management System
FIFO	First In First Out
JVM	Java Virtual Machine
LIFO	Last In First Out
OSIRIS-SE	Open Service Infrastructure for Reliable and Integrated Process Support – Stream Enabled [BSS06, Bre08], ein Datenstromsystem
PNA	Positive-negative approach (Positiv-Negativ-Ansatz)
RMS	Effektivwert, von engl. 'root mean square'
SBDS	Schwankungsbeschränkter periodischer Datenstrom
SZT	Sofortiger Zustandstransfer
TIA	Time-interval approach (Zeitintervallansatz)
ZT	Zustandstransfer
ZTS	Zustandstransferstrategie

Migrationsstrategien

BSC	Background State Computation [YKPS07], Erweiterung für HybMig
GenMig	General Migration [Krä07]
GHM	Generalized Hybrid Migration [YKPS07]
GMS	Generalized Moving States [YKPS07]
GPT	Generalized Parallel Track [YKPS07]
HybMig	Hybrid Migration [YKPS07]
MS	Moving State Strategy [Zhu06]
PT	Parallel Track Strategy [Zhu06]

Zustandstransferstrategien

DS	Distributed Selection
LF	Latest First
OF	Oldest First
RS	Random Selection
RSD	Random Selection without Doubles
SemS	Semantic Selection

Formelzeichen

δ	Schrittweite eines Fensters
$\mathbb{N}^+$	Menge der natürlichen Zahlen ohne 0
$\mathbb{N}$	Menge der natürlichen Zahlen mit 0
$\mathbb{R}$	Menge der reellen Zahlen
$\mathcal{A}$	Menge von Attributen eines Datenstromelements
$\mathcal{D} = \langle \mathcal{A}, ts \rangle$	Datenstromelement $\mathcal{D}$ bestehend aus Datentupel $\mathcal{A}$ und Zeitstempel ts
$\mathcal{N}_a$	Summe transferierbarer Zustandswerte am Anfang der Migration
$\mathcal{N}_n$	Summe neuer Werte aller Zustände
$\mathcal{N}_o$	Summe Zustandstransferwerte aller Zustände
$\mathcal{Q}$	Warteschlange
$\mathcal{S}_I$	Eingangsdatenstrom
$\mathcal{S}_O$	Ausgangsdatenstrom
$\mathcal{S}_Z$	Zustandstransferdatenstrom
$\mathcal{S}$	Datenstrom
$\mathcal{Z}$	Zustand
$\overline{N}_i$	Mittlere Anzahl eingehender Werte pro Zyklus
$\overline{T}_p^*$	Durchschnittliche Verarbeitungszeit mehrerer Eingangswerte
$\overline{T}_i$	Durchschnittliche Pausendauer
$\overline{T}_{syst}$	Mittlerer Ereignisabstand eines aperiodischen Systems
$\overline{T}$	Mittlerer Ereignisabstand
τ	Abweichung eines schwankungsbeschränkten Datenstroms [Ham05]
A_i	Attribut eines Datentupels
a_i	Attributwert
$bias$	Systematische Abweichung
d_{DS}	Abstand zweier Zustandstransferwerte beim Transfer mit DS
D_{syst}	Mindestabstand eines schwankungsbeschränkten Systems (mehrere Eingangsdatenströme)
D	Mindestabstand eines schwankungsbeschränkten Datenstroms [Ham05]
d	Abstand zweier Eingangsdatenströme (Reihenfolge)
N_a	Anzahl transferierbarer Zustandswerte am Anfang der Migration
N_c	Anzahl notwendiger Zyklen für den Zustandstransfer
N_e	Anzahl der Eingangsdatenströme einer Anfrage

N_g	Gruppengröße für den Transfer eines Zustandes
N_i	Anzahl eingehender Werte pro Zyklus
N_m	Anzahl notwendiger Zyklen für die Migration
N_n	Anzahl neuer Werte eines Zustandes
N_o	Anzahl Zustandstransferwerte eines Zustands
N_p	Anzahl der verbleibenden Werte während einer Pause
N_r	Anzahl restlicher Werte eines Zustandstransfers
N_t	Anzahl transferierbarer Werte während einer Pause
N_z	Anzahl der Zustände in der Zustandspaarmenge
$N_{o,r}$	Anzahl der verbleibenden Werte eines Zustands während des Zustandstransfers
Op	Operator
r	Reserve
sel	Selektivität
T_p^*	Verarbeitungszeit mehrerer Eingangswerte
T_A	Anlaufzeit
T_a	Anfangszeit
T_i	Pausendauer
$t_{M,end}$	Zeitpunkt des Migrationsendes
$t_{M,start}$	Zeitpunkt des Migrationsbeginns
T_M	Migrationsdauer
T_{OS}	Ergebnisverzögerung (Output Silence)
T_p	Verarbeitungszeit eines Eingangswertes
T_{syst}	Periodendauer eines periodischen Systems (mehrere Eingangsdatenströme)
T_S	Umschaltdauer
T_t	Tupeltransferzeit
T_v	Zeitliche Verschiebung von Eingangsdatenströmen
T_{ZT}	Zustandstransferdauer
ts_{max}	Maximaler Zeitstempel
ts_{min}	Minimaler Zeitstempel
ts	Zeitstempel
T	Periodendauer
w^*	Fenstergröße (zeitbasiert)
W_n	Menge von Zustandswerten aus mehreren Zuständen
w_n	Neue Fenstergröße
w_o	Originale Fenstergröße
w	Fenstergröße
ZTS	Zustandstransferstrategie

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

Datenstromsysteme ermöglichen die schnelle Verarbeitung von großen Datenmengen, da die zu verarbeitenden Daten nur im Arbeitsspeicher gehalten werden. Die Verarbeitungslogik wird in Form von kontinuierlichen Anfragen ausgeführt, welche aus Operatoren modelliert sind. Operatoren führen Operationen auf den durchströmenden Daten aus, z. B. Filtern, Aggregieren oder Kombinieren. Manche Operatoren verwenden dafür vergangene Daten, welche sie in inneren Zuständen speichern. Aufgrund der Verarbeitung im Arbeitsspeicher sind die inneren Zustände flüchtig. Ein persistentes Abspeichern von inneren Zuständen oder Zwischenwerten der Verarbeitung kommt nicht in Frage, da Schreiboperationen auf konventionellen Datenträgern gegenwärtig ca. einhundert Mal langsamer sind als im Arbeitsspeicher.

1.1 Problemstellung

Im Laufe der Zeit können wechselnde Bedingungen Änderungen der Anfragen erfordern, z. B. zusätzliche Operatoren nach einem Sensorwechsel oder die Verwendung eines optimierten Algorithmus oder Anfragenaufbaus. Dementsprechend sind die kontinuierlichen Anfragen während ihrer Laufzeit zu modifizieren. Die Überführung einer Anfrage in eine aktualisierte Version wird Migration genannt. In Systemen mit Echtzeitanforderungen und hohen Datendurchsätzen bedarf es einer speziellen Migrationsmethodik, um Änderungen ohne negativen Einfluss auf das laufende System zu integrieren. Die größte Herausforderung stellt dabei der innere Zustand einer Anfrage im Arbeitsspeicher dar, der bei einem Neustart der Anfrage verloren ginge. Zudem ist eine Unterbrechung der Verarbeitung und Ergebnisausgabe in vielen Anwendungsfällen nicht akzeptabel.

Eine Migration ist deshalb parallel zur regulären Verarbeitung durchzuführen, um stets Ergebnisse ausgeben zu können. Zu einem geeigneten Zeitpunkt kann dann zur aktualisierten Version der Anfrage gewechselt werden. Um eine optimale Migration zu ermöglichen, ist der innere Zustand der Verarbeitungslogik zu erhalten. Dabei ist zu beachten, dass Operatoren unter Umständen auf andere Bearbeitungsknoten verlagert wurden. Der Transfer eines Zustandswertes zum neuen Operator ist dann über das Netzwerk zu realisieren und benötigt Zeit.

Für verschiedene Systeme wurden Migrationsstrategien vorgestellt, welche jedoch die

Anforderungen nicht gleichzeitig erfüllen können. Existierende Ansätze zur Anfragenmigration sind entweder langsam, da sie auf einen Zustandstransfer verzichten oder sie unterbrechen die Ergebnisberechnung und -ausgabe während der Migration. Andere sind nicht in verteilten Umgebungen anwendbar oder übertragen Zustandswerte, die nie zu einem Ergebnis beitragen.

Durch die Beschränkung des Zustandstransfers auf notwendige Zustandswerte kann die Migration beschleunigt werden. Das Ziel ist eine schnelle Migration und gleichzeitig eine garantierte Datenverarbeitung und Ergebnisausgabe.

1.2 Beitrag der Arbeit

Diese Arbeit präsentiert umfassende Lösungen für die Anpassung von kontinuierlichen Datenstromanfragen während der Laufzeit aufbauend auf bestehenden Migrationsprinzipien. Dabei sind die folgenden Beiträge hervorzuheben.

1. Ein allgemeines Vorgehensmodell zur Laufzeitanpassung von kontinuierlichen Datenstromanfragen – Duplikation, Modifikation, Reintegration.
2. Mehrere Zustandstransferstrategien mit unterschiedlichen Transfereigenschaften für verschiedene Anforderungen und Randbedingungen zur Minimierung der Transfermenge und somit zur Reduktion der Transferlast.
3. Ein numerisches Modell zur Berechnung der Migrationskonfiguration in periodischen Datenströmen, einschließlich Transferdauer- und Migrationsdauervorhersage.
4. Basierend auf 3., eine Heuristik für die Anwendung in aperiodischen Datenströmen.

1.3 Aufbau der Arbeit

Im nachfolgenden Kapitel 2 werden nach einem einführenden Beispiel die notwendigen Grundlagen erklärt. Insbesondere wird auf zustandsbehaftete Operatoren, deren Anlaufzeit und das Anpassen von Datenstromanfragen eingegangen. Das Kapitel wird mit der Aufstellung von Anforderungen an die Migration abgeschlossen.

Existierende Arbeiten werden in Kapitel 3 analysiert und diskutiert sowie deren Stärken und Schwächen aufgezeigt. Dabei werden nicht nur Migrationsverfahren in Datenstromsystemen berücksichtigt, sondern auch relevante Ansätze, wie verteilte Systeme oder Parallelisierung von Operatoren, auf anwendbare Konzepte untersucht.

In Kapitel 4 wird ein allgemeines Vorgehensmodell zur Anpassung von kontinuierlichen Datenstromanfragen vorgestellt. Das Vorgehensmodell setzt sich aus drei Schritten zusammen – Duplikation, Modifikation, Reintegration. Die Duplikation dient der Erstellung von entkoppelten Testanfragen, die während der Modifikation verändert werden.

Das Ergebnis ist eine aktualisierte Anfrage. Mit der Reintegration wird die aktualisierte Anfrage in das laufende System überführt, um mittels Migration die ursprüngliche Anfrage zu ersetzen. Eine besondere Rolle nimmt dabei der Zustandstransfer ein.

Der Zustandstransfer wird in Kapitel 5 für periodische Datenströme und Anwendungsfälle mit verschiedenen Eigenschaften analysiert. Es wird ein Verfahren zum Ermitteln einer geeigneten Konfiguration für eine Migration mit Zustandstransfer und ein entsprechendes numerisches Modell zur Berechnung der Migrationsparameter vorgestellt. Von besonderer Bedeutung sind dabei die Zustandstransferstrategien. Schließlich werden für die Durchführung der Migration und des Zustandstransfers notwendige funktionale Erweiterungen des Datenstromsystems beschrieben.

Für aperiodische Datenströme wird in Kapitel 6 eine einfache Heuristik präsentiert, welche auf den Erkenntnissen des vorigen Kapitels aufbaut.

In Kapitel 7 werden zunächst einige Details zur prototypischen Implementierung der Migrations- und Zustandstransferkonzepte dargestellt. Des Weiteren wird mit experimentellen und simulativen Analysen die Funktionsfähigkeit der vorgestellten Konzepte nachgewiesen, deren Eigenschaften werden veranschaulicht und Vorteile im Vergleich zu existierenden Ansätzen gezeigt.

Abschließend wird die Arbeit in Kapitel 8 zusammengefasst und ein Ausblick auf zukünftige Forschungsmöglichkeiten und Ansätze zur Weiterentwicklung gegeben.

Kapitel 2

Datenstromverarbeitung

Mit dem Bedarf, große Datenmengen in Echtzeit zu verarbeiten, stoßen Datenbanksysteme an technische Grenzen, vor allem in Bezug auf Zugriffszeiten bei Schreiboperationen, die in Datenbanksystemen vor der Datenverarbeitung stattfinden müssen. So wurden um das Jahr 2000 die ersten Datenstromsysteme entwickelt, darunter STREAM [ABB+04], Aurora [BBC+04] und TelegraphCQ [CCD+03].

Im Unterschied zu Datenbanken werden in Datenstrommanagementsystemen (engl. „data stream management system“, DSMS) die eingehenden Daten in Form von Eingangsdatenströmen direkt im Arbeitsspeicher durch kontinuierliche Anfragen verarbeitet, wobei Daten unter Umständen in inneren Operatorzuständen ebenfalls im Arbeitsspeicher gepuffert werden. Die Ergebnisse der Verarbeitung werden als Datenströme ausgegeben. Eine Datenbank (DB) wird lediglich zum Bereitstellen der Anfragen benötigt und speichert bei Bedarf Ergebnisse der Verarbeitung. Es wird somit das Prinzip der Datenbanksysteme umgekehrt – in DSMS sind Daten dynamisch und Anfragen statisch. In Abbildung 2.1 ist das Prinzip der Datenstromverarbeitung (nach [SÇZ05]) dargestellt. DSMS können als zentrales System oder verteilt realisiert sein.

Beispielhafte Anwendungen von DSMS, etwa für die Beobachtung von Börsenkursen, zur Sensordatenverarbeitung oder zur Straßenverkehrsüberwachung, wurden u. a. in [GÖ03a] beschrieben. Für verteilte Datenerfassung und -verarbeitung werden beispielsweise Sensornetzwerke genutzt – auch unter Verwendung von Datenströmen [GG07]. Am Beispiel der Zustandsüberwachung von Produktionsmaschinen zum Zweck der zustandsorientierten Instandhaltung werden eine mögliche Anwendung eines verteilten Datenstromsystems sowie resultierende Herausforderungen kurz diskutiert.

2.1 Motivation

Zustandsorientierte Instandhaltung ist eine präventive Instandhaltungsstrategie [RF10]. Es gilt in erster Linie, Ausfälle durch zu späte Wartung und damit verbundene Stillstandszeiten zu verhindern. Zu zeitige Wartungen sind jedoch zu vermeiden, um die Restlaufzeit von Anlagen optimal zu nutzen. Mit dem Ziel einer Optimierung der Wartungsplanung werden Abnutzungsvorgänge bei Maschinen und Anlagen durch die Erfassung von zahlreichen Messgrößen beobachtet. Da diese Strategie im Vergleich zu anderen

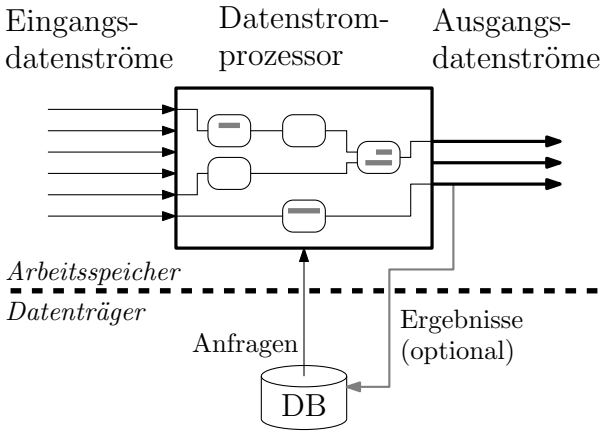


Abbildung 2.1: Prinzip der Datenstromverarbeitung (nach [SÇZ05])

Instandhaltungsstrategien mit einem größeren Aufwand zu implementieren ist, wird sie vor allem in Szenarien eingesetzt, in denen Wartung und Ausfallzeiten teuer sind. Um hinreichend Zeit für das Planen und Auslösen von Wartungsaufträgen und für die logistischen Prozesse zur Verfügung zu haben, ist eine rechtzeitige Diagnose oder Prognose von Fehlern notwendig. Zum Erreichen einer Produktion ohne ungeplante Stillstände sind prognostische Verfahren deutlich effizienter [JLB06, WDD94].

Voraussetzung für die Einschätzung des Anlagenzustands und die Vorhersage von Restlaufzeiten ist eine kontinuierliche Zustandsüberwachung. Üblicherweise werden kritische Anlagenteile überwacht, beispielsweise Getriebe, Lager oder Wellen. Daten von laufenden Maschinen werden in Form von diskreten Signalen von Maschinen erfasst und verarbeitet, z. B. Spannungen, Ströme, Temperaturen, Vibrationen, Drücke, akustische Daten oder Messgrößen aus der Umgebung der Maschine. Die erfassten Daten werden im Zeitbereich oder im Frequenzbereich analysiert um sogenannte Merkmale zu extrahieren. Beispiele für Merkmale des Zeitbereichs sind der Effektivwert (RMS, von engl. „root mean square“), die Standardabweichung, die Schwingungsbreite („peak-to-peak“), der Scheitelfaktor (Crest-Faktor) oder Korrelationskoeffizienten. Merkmale des Frequenzbereichs werden beispielsweise aus Spektren der Fourier-transformierten Signale extrahiert. Es ist sowohl im Zeitbereich als auch im Frequenzbereich möglich, Signale durch Mitteln, Filtern, Rauschreduktion und andere Methoden vorzuverarbeiten, um deren Qualität zu erhöhen. Die gewonnenen Merkmale oder Kombinationen von ihnen werden schließlich auf Modelle angewandt, um die kurz- oder langfristige Entwicklung des Anlagenzustands zu beobachten und die verbleibende Lebensdauer abzuschätzen [JLB06, Sch10].

Fallstudien und experimentelle Untersuchungen zur Abnutzung bis hin zur Zerstörung von Anlagenkomponenten [QLLY03, WRBK01] haben gezeigt, dass es äußerst schwierig ist, deterministische Modelle zu entwickeln und den Verlauf der Abnutzung schon von Beginn ihres Auftretens vorherzusagen. Sogar unter gleichen Versuchsbedingungen konnte für gleiche Merkmale unterschiedliche Verschleißmuster gezeigt werden. Bei ei-

nigen Versuchen war der Anstieg der Abnutzung so groß, dass nur sehr wenig Zeit für Wartungsmaßnahmen bleiben würde.

In der Literatur wird die Kombination von Merkmalen als wirkungsvoller Ansatz zur Zustandsbestimmung von Anlagenteilen und zur Vorhersage der verbleibenden Lebensdauer diskutiert. Allerdings ist die Merkmalskombination nicht trivial. Besondere Herausforderungen sind die Auswahl von Merkmalen und die Entwicklung von geeigneten Vorhersagemodellen. Beide Arbeitsschritte werden gewöhnlich während der Konzeptphase, z. B. von Entscheidungsunterstützungssystemen, durchgeführt [Car05]. Obwohl es einen steigenden Bedarf für Zustandsüberwachung und zustandsorientierte Instandhaltung gibt, sind gerade die Modellentwicklung und häufige Konzeptänderungen die Gründe, warum eine Implementierung im industriellen Umfeld sehr schwer fällt [JLB06].

Für die Anwendung von Zustandsüberwachung und zustandsorientierte Instandhaltung lassen sich Herausforderungen ableiten, welche nachfolgend diskutiert werden, unter dem Gesichtspunkt, dass ein Datenstromsystem für die Verarbeitung eingesetzt wird. Es sei daran erinnert, dass Operatoren und Anfragen, welche die einzelnen Verarbeitungsschritte, wie Merkmalsberechnung, Merkmalskombination oder Vorhersage realisieren, im Arbeitsspeicher ausgeführt werden.

- *Kontinuierliche Überwachung:* Kritische Maschinen und veränderliches Signalverhalten erfordern eine durchgehende Überwachung, um Merkmale aus der Vielzahl von erfassten Daten und Signalen zu extrahieren. Für viele Merkmale müssen vergangene Werte berücksichtigt werden, z. B. Zeitreihen, mehrere Perioden oder Daten der letzten Arbeitsstunden verteilt über mehrere Tage. Eine schnelle Verarbeitung wird ebenso benötigt, idealerweise in einem einzigen Durchlauf über die Daten, d. h. ankommende Daten werden nur einmal vom verarbeitenden Algorithmus ausgewertet.

- *Echtzeitausgabe:* Damit angeschlossene Systeme rechtzeitig reagieren können, ist eine Verarbeitung und Ausgabe in Echtzeit notwendig. Dies schließt eine garantierte Ausgabe von Ergebnissen ein. Eine Verzögerung in der Berechnung von einzelnen Merkmalen resultiert in der Verzögerung der nachfolgenden Verarbeitungsphasen, also der Kombination von Merkmalen und der Vorhersageverfahren.

- *Anpassung während der Laufzeit:* Die Anforderungen bezüglich der Datenerfassung und -verarbeitung können sich während der Laufzeit ändern, insbesondere bei einer langfristigen Anwendung eines Systems. So können beispielsweise defekte Sensoren ersetzt oder neue hinzugefügt werden. Auch die Charakteristik der Sensorik, von Maschinen oder des Umfeldes können sich über die Zeit ändern. Außerdem können Modelländerungen notwendig sein, um die Effizienz der Datenverarbeitung und der Vorhersagen zu erhöhen. Derartige Änderungen müssen sehr wahrscheinlich in der Verarbeitungslogik abgebildet werden, weshalb ein anpassbares Verarbeitungssystem notwendig ist. Aufgrund der ersten zwei Anforderungen müssen Änderungen

während der Laufzeit integriert werden, ohne den operativen Betrieb zu stören, zu verlangsamen oder gar zu unterbrechen.

- *Zustandserhaltung:* Sofern vergangene Werte für die Berechnung verwendet werden, sind diese im Arbeitsspeicher flüchtig gespeichert. Diese Daten können im Laufe der Anpassung entweder verworfen oder erhalten werden. Nach dem Verwerfen ist mit einer erhöhten Übergangszeit zu rechnen, da die Fenster kontinuierlich mit ankommenden Werten gefüllt werden müssen. Eine Zustandserhaltung kann die Übergangszeit wesentlich verkürzen.

- *Verteiltes System:* Da die Messpunkte der zu erfassenden Signale räumlich voneinander entfernt sein können, ist die Verarbeitung durch ein verteiltes System, z. B. ein Sensornetzwerk, sinnvoll, da so die lokal verfügbaren Ressourcen genutzt werden, beispielsweise für eine Aggregation nahe der Datenquelle, um Netzwerklast und Übertragungszeiten zu reduzieren. Dies stellt aber auch eine besondere Herausforderung im Zusammenhang mit der Zustandserhaltung dar, da bei einer Zustandserhaltung beispielsweise Transferzeiten zu berücksichtigen sind.

- *Testen von Alternativen:* Für Benutzer ist das Erstellen und Evaluieren von neuen oder modifizierten Anfragen in einer Testumgebung eine wesentliche Unterstützung im Entwicklungs- und Optimierungsprozess. Besonders die Verwendung von Daten aus dem produktiven System, aber auch von historischen Daten oder Simulationsdaten, erlauben zahlreiche Variationsmöglichkeiten beim Vergleich von Alternativen. Eine einfache Übertragung der optimalen, modifizierten Anfrage in das produktive System sollte gewährleistet sein. Trotzdem ist eine geeignete Entkoppelung der Testumgebung notwendig, um das produktive System während der Tests nicht zu beeinflussen.

Besonders die ersten zwei Herausforderungen, kontinuierliche Überwachung und Echtzeitausgabe, werden oft als Grundeigenschaften von DSMS genannt, weshalb sich solche Systeme für die Realisierung anbieten. Gerade die sofortige Verarbeitung im Arbeitsspeicher und der Verzicht, Rohdaten vor der Verarbeitung auf den wesentlich langsameren Datenträgern zu speichern, tragen zur Erfüllung dieser Anforderungen bei. Mit der Anforderung einer langfristigen Überwachung resultiert jedoch ein Konflikt zwischen einer garantierten Ergebnisausgabe und den durchzuführenden Systemanpassungen, insbesondere dadurch, dass die Daten im Arbeitsspeicher flüchtig sind und bei einem Neustart verloren gehen. Dieser Konflikt ist mit geeigneten Konzepten zu lösen.

2.2 Grundlagen

2.2.1 Datenströme und Datenstromelemente

Ein Datenstrom $\mathcal{S}$ ist eine potentiell unendliche Sequenz von Datenstromelementen. Jedes Datenstromelement $\mathcal{D} = \langle \mathcal{A}, ts \rangle$ besteht aus einer Menge von Attributen $\mathcal{A}$ und einem Zeitstempel ts. Die Attributmenge $\mathcal{A}$ eines Datenstromelements – auch Datentupel oder Tupel genannt – enthält eine endliche Menge von n Attributen A_i $(1 \leq i \leq n)$, $\forall i, n \in \mathbb{N}^{+}$[1], welche einem Schema entsprechen. Jedes Attribut A_i besitzt einen Attributwert a_i. Beispiele für einfache Attributwerte sind Identifikationsnummern, Zähler- oder Messwerte. Diese können direkt für die Verarbeitung verwendet werden. Komplexe Attributwerte bedürfen meistens einer Extraktion oder einer Vorverarbeitung, beispielsweise Zeitangaben mit Datum und Uhrzeit, aus denen ein Teil des Datums verwendet werden soll, oder XML-Fragmente eines XML-Datenstroms [GGM+04], welche vor der Verarbeitung geparst werden müssen.

Der Zeitstempel ts des Datenstromelements wird vom DSMS vergeben und ist somit nach [BBD+02] implizit. Der implizite Zeitstempel dient unter anderem zum Sortieren der Datenstromelemente. Statt eines Zeitstempels können auch andere Sortiermerkmale benutzt werden, z. B. ein inkrementeller Zähler. Außerdem kann ein Datenstromelement explizite Zeitstempel als Bestandteil des Datentupels enthalten. Diese werden jedoch nicht weiter betrachtet. Alle weiteren Betrachtungen beziehen sich, wie auch in anderen Datenstromsystemen, auf die impliziten Zeitstempel[2].

Datenstromelemente können einfache oder zusammengesetzte Zeitstempel besitzen. Datenstromelemente mit einem einfachen Zeitstempel („singleton timestamp“ [Zhu06]) werden als Basiselemente oder Basistupel bezeichnet, da sie ihren Zeitstempel bei Ankunft im System erhalten und ausschließlich diesen besitzen, sieht man von expliziten Zeitstempeln ab. Basistupel sind in jedem Fall alle unveränderten Datenstromelemente aber auch gefilterte Basistupel, oder solche Datenstromwerte, deren Attribute verändert wurden aber nicht deren Zeitstempel. Im Gegensatz dazu entstehen zusammengesetzte Zeitstempel („combined timestamp“ [Zhu06]) während der Verarbeitung in der Anfrage durch Kombination von Datenstromwerten, z. B. durch Verbundoperatoren, oder durch Modifikationen an Datenstromwerten, die auch den Zeitstempel betreffen. Insbesondere Verbundoperatoren setzen Attribute und Zeitstempel von Datenstromwerten verschiedener Ströme zusammen. Die Bestandteile werden als Sub-Tupel bezeichnet. Die Zeitstempel der Sub-Tupel werden im Zeitstempel des Ergebnistupels vereinigt, so dass $ts = \{ts_1, \ldots, ts_m\}, \forall m \in \mathbb{N}^{+}$. So besteht die Möglichkeit, alle Zeitstempel mitzuführen, selbst wenn zusammengesetzte Datenstromwerte kombiniert werden. Mit

[1] Es wird $\mathbb{N}^{+}$ für die Menge der natürlichen Zahlen ohne 0 verwendet und $\mathbb{N}$ entsprechend mit 0. $\mathbb{R}$ bezeichnet die Menge der reellen Zahlen.

[2] Ergänzend dazu sind der Zeitintervallansatz und der Positiv-Negativ-Ansatz zu nennen, welche z. B. im System PIPES [KS04] verwendet werden. Ihre Semantik und die Auswirkungen auf die Migration werden an entsprechender Stelle beschrieben.

$ts_{max} = \max(ts)$ kann für einen Zeitstempel der größte aller Sub-Tupel-Zeitstempel ermittelt werden, also der neueste aller Zeitstempel eines Datenstromelements[3]; mit $ts_{min} = \min(ts)$ analog der kleinste bzw. der älteste Zeitstempel. Für Basiselemente gilt $ts = ts_{min} = ts_{max}$.

Datenströme können entweder durch externe Datenquellen generiert und an das DSMS gesendet werden oder das DSMS erzeugt einen Datenstrom selbst durch Abfragen von externen Quellen, z. B. Sensoren. Außerdem ist es bei manchen DSMS möglich, Datenströme durch Simulationsalgorithmen zu generieren oder archivierte Datenströme zu laden, um diese erneut ablaufen zu lassen. Datenstromelemente, die für die Verarbeitung nicht mehr relevant sind, werden verworfen oder archiviert. Sie stehen danach nicht mehr für die Verarbeitung zur Verfügung bzw. müssen zeitaufwendig aus dem Archiv geladen werden, beispielsweise aus einer Datenbank.

Typischerweise besitzen Datenströme einen hohen Durchsatz, d. h. es werden sehr viele Nachrichten pro Zeiteinheit erzeugt. Zudem wird eine geringe Verzögerung zwischen ankommenden Daten und der Ausgabe von Ergebnissen erwartet. Zum Beispiel stellt ein Dienstleistungsunternehmen (OPRA, Options Price Reporting Authority) Börsendaten mit Hilfe von Datenströmen in Echtzeit zur Verfügung. Für 2012 wurden Anforderungen mit Spitzenwerten von über zehn Millionen Nachrichten pro Sekunde definiert. Dies entspricht zugleich einem Datenvolumen von über 300 MB pro Sekunde [Cor12]. Die akzeptable Verzögerung für Online-Börsenhandel liegt unter einer Sekunde [SÇZ05]. Datenstromanwendungen müssen also große Datenmengen verarbeiten und gleichzeitig eine geringe Verzögerung garantieren. Auch in anderen Bereichen, z. B. in industriellen Anwendungen, sind in Zukunft ähnliche Anforderungen zu erwarten [GB12].

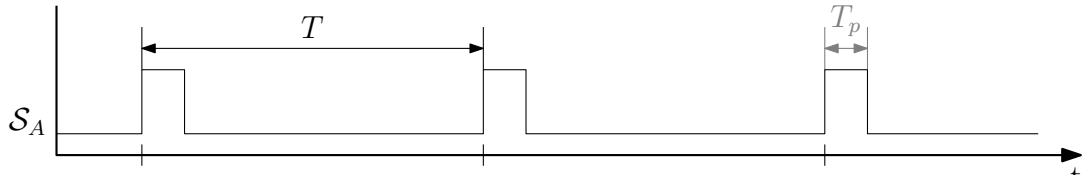
Datenströme haben im Allgemeinen eine variable Ankunftsrate, da ein DSMS keine Kontrolle über externe Datenquellen besitzt und Daten dann empfängt, wenn diese von der Quelle erzeugt und versandt wurden, beispielsweise durch Send-on-Delta-Verfahren [Mis06]. Dazu kommen Verzögerungen durch die Übertragung im Netzwerk. Ebenso entstehen variable Ankunftsraten durch aperiodisches Abtasten durch das DSMS. All diese Varianten lassen sich zusammenfassen als aperiodische Datenströme.

Verschiedene Datenstromtypen und deren typische Parameter sind in Abbildung 2.2 als Impulsdiagramm dargestellt, wobei eine steigende Flanke das Auftreten eines Ereignisses kennzeichnet und eine fallende Flanke das Ende seiner Verarbeitung. Die Länge eines Impulses entspricht demzufolge der Verarbeitungszeit T_p eines Ereignisses.

Eine konstante Ankunftsrate kann durch zyklisches Senden oder Abfragen erzeugt werden. Der zeitliche Abstand zwischen aufeinanderfolgenden Datenstromelementen ist dann immer gleich. Der Datenstrom besitzt eine konstante Periodendauer T (Abbildung 2.2a). Ein solcher periodischer Datenstrom[4] stellt jedoch einen Sonderfall dar. In realen Systemen kann bestenfalls von schwankungsbeschränkten periodischen Datenströmen ausgegangen werden.

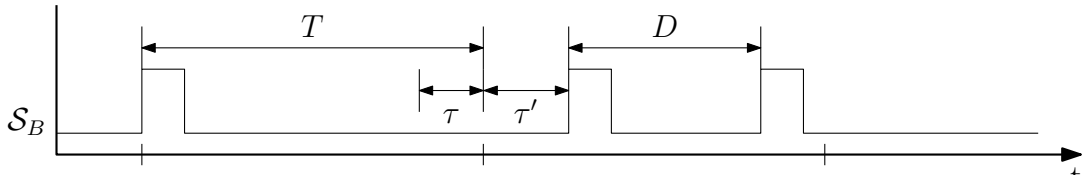
[3] Sofern nicht anders beschrieben, ist mit ts immer ts_{max} gemeint.
[4] oder auch äquidistanter Datenstrom

(a) periodischer Datenstrom



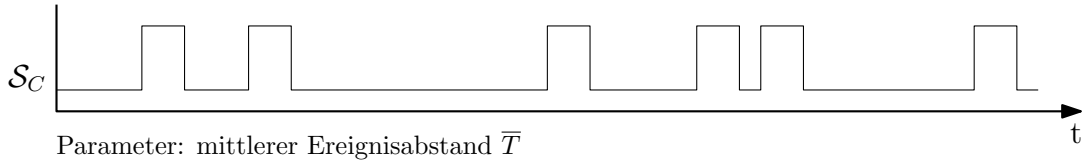
Parameter: Periodendauer T

(b) schwankungsbeschränkter Datenstrom (nach [Ham05], vereinfacht)



Parameter: theoretische Periodendauer T, Mindestabstand D, Schwankung (τ, τ')

(c) aperiodischer Datenstrom



Parameter: mittlerer Ereignisabstand $\overline{T}$

Abbildung 2.2: Datenstromtypen

In [Ham97] wurde ein allgemeines Modell für diesen Datenstromtyp (engl. „jitter constrained periodic stream“) definiert und mit gleichartigen Ansätzen verglichen. Laut dieser Definition ist ein „(τ, τ')-schwankungsbeschränkter Strom mit konstanter Rate $R = T^{-1}$ und Mindestabstand D (kurz: schwankungsbeschränkter Strom)“ [Ham05] eine Folge von Ereignissen, deren tatsächliche Ereigniszeit (entspricht dem Zeitstempel ts) im Intervall $[iT - \tau, iT + \tau']$, $\forall i \in \mathbb{N}$ liegt und deren Abstände nicht kleiner sind als D, d. h. $ts_{i+1} - ts_i \geq D$, $\forall i \in \mathbb{N}$. Es sind $T, D, \tau, \tau' \in \mathbb{R}$ mit $T > D > 0$ und $\tau, \tau' > 0$. Dabei charakterisiert T die theoretische Periodendauer und somit iT die theoretische Ereigniszeit des i-ten Ereignisses. Mit D wird der Mindestabstand von aufeinanderfolgenden Ereignissen in der oben beschriebenen Form festgelegt. Die Parameter τ und τ' sind die potentiellen zeitlichen Abweichungen von der theoretischen Ereigniszeit, beschreiben also, ob ein Ereignis verfrüht oder verspätet eintritt. Im Modell sind beide Abweichungen unabhängig von den anderen Größen und können sogar größer als T sein. Das definierte Modell wird in [Ham97] und [Ham05] hauptsächlich für Datenströme verwendet, in denen Nachrichtenhäufungen (sogenannte Bursts) auftreten können. Insbesondere werden verschiedene Burst-Szenarien diskutiert.

Eine gesonderte Handhabung von Burst-Verhalten ist nicht Gegenstand dieser Arbeit, da Burst-Ströme wie aperiodische Datenströme behandelt werden können. Das Modell für schwankungsbeschränkte Datenströme ist dennoch wichtig, um periodische Datenströme mit geringen Abweichungen zu beschreiben. Es werden in dieser Arbeit konkret Datenströme betrachtet, welche die Grundordnung der Datenstromwerte erhalten und deren Abweichungen nur innerhalb einer Periode schwanken. Unter Berücksichtigung der Verarbeitungszeit gilt folgende Bedingung:

$$\tau + \tau' < T - T_p \,. \tag{2.1}$$

Der Mindestabstand D von aufeinanderfolgenden Ereignissen kann auf Grund der Vereinfachung aus den anderen Parametern des schwankungsbeschränkten Datenstroms errechnet werden. Nimmt man den ungünstigsten Fall an, dann ist

$$D = T - \tau - \tau'. \tag{2.2}$$

Ein beispielhaftes Schema des im weiteren Verlauf verwendeten, vereinfachten Modells ist in Abbildung 2.2b dargestellt.

Für aperiodische Datenströme kann eine genäherte Größe aus den Ereigniszeiten berechnet werden. Dafür wird der Abstand von aufeinander folgenden Ereignissen gemittelt. Es sei Δts der Abstand zweier aufeinander folgenden Ereignisse, d. h. $\Delta ts_{i+1} = ts_{i+1} - ts_i$. Der mittlere Ereignisabstand $\overline{T}$ ist dann der Mittelwert aller Ereignisabstände, $\overline{T} = 1/i * \sum_1^i \Delta ts_{i+1}$. In Abbildung 2.2c repräsentiert $\mathcal{S}_C$ einen aperiodischen Datenstrom.

Die Eigenschaften von Datenströmen können über die Zeit variieren. Neben den bereits diskutierten Nachrichtenhäufungen besteht die Möglichkeit, dass Datenströme unterbrochen werden. Unterbrechungen sind häufig die Ursache für anschließende Bursts.

Des Weiteren können Datenstromelemente verloren gehen oder Fehler aufweisen. Eine weitere Schwierigkeit stellt eine vertauschte Reihenfolge von Datenstromelementen dar, beispielsweise eine temporale Vertauschung, d. h. die zeitliche Reihenfolge der Datenstromelemente ist nicht monoton steigend, ein Effekt, der beispielsweise durch den Datenaustausch in nichtdeterministischen Netzwerken entstehen kann. Langfristig können Probleme wie Drift[5] oder Rauschen auftreten. Variierende Eigenschaften machen es schwierig, das Verhalten eines Datenstroms vorherzusagen und eine Migrationsstrategie optimal einzusetzen.

2.2.2 Verarbeitung durch Anfragen

Datenströme werden durch Anfragen verarbeitet. Dabei können nach [BBD+02] Anfragetypen bezüglich der Häufigkeit der Ausführung und des Zeitpunktes der Erstellung der Anfrage unterschieden werden. Im Hinblick auf die Ausführungshäufigkeit trennt man zwischen einmaligen Anfragen und kontinuierlichen Anfragen. Einmalige Anfragen werden nur einmal auf einem aktuellen Datensatz verarbeitet und entsprechen somit dem Prinzip einer Abfrage eines relationalen Datenbanksystems. Im Gegensatz dazu werden kontinuierliche Anfragen fortlaufend auf den durchströmenden Daten ausgeführt. Dabei werden alle Daten berücksichtigt, die noch nicht verworfen wurden. Die Ergebnisse werden kontinuierlich ausgegeben und bilden somit ebenfalls einen Datenstrom. Hinsichtlich des Erstellzeitpunktes lässt sich zwischen vordefinierten Anfragen und Ad-hoc-Anfragen unterscheiden. Vordefinierte Anfragen befinden sich bereits im System und haben alle benötigten Daten für die Berechnung zur Verfügung. Das heißt die Anfrage befindet sich in einem stationären Zustand. Dies gilt nicht für Ad-hoc-Anfragen. Eine Ad-hoc-Anfrage wird zur Laufzeit an das System übermittelt und benötigt unter Umständen Daten, die bereits in der Vergangenheit erzeugt wurden und evtl. schon verworfen wurden. Da ein exaktes Ergebnis erst im stationären Zustand berechnet werden kann, muss eine Anlaufzeit für solche Anfragen berücksichtigt werden, bis der stationäre Zustand erreicht ist. Während der Anlaufzeit kann aufgrund fehlender Daten meistens nur ein genähertes Ergebnis ermittelt werden. Sind ausschließlich exakte Ergebnisse zulässig, kann dann bis zum Ende der Anlaufzeit kein Ergebnis ausgegeben werden.

2.2.3 Systemelemente von Datenstromanfragen

Datenstromanfragen sind in den meisten Systemen als Baum aufgebaut und bestehen aus drei Typen von Systemelementen: Datenquellen, Operatoren und Datensenken. Eine Datenquelle repräsentiert eine externe Datenquelle, z. B. einen Sensor oder eine andere Anwendung. Die Datenquelle bildet somit den Zugang ins DSMS für einen oder mehrere Datenströme. Sie hält eine Verbindung zur externen Quelle und erhält darüber Daten oder fragt Daten darüber ab. Neue Daten werden von der Datenquelle in Datenstrom-

[5] langsame, ungewollte Änderung eines Messsignals, z. B. durch Alterung der Sensorik [Sch07]

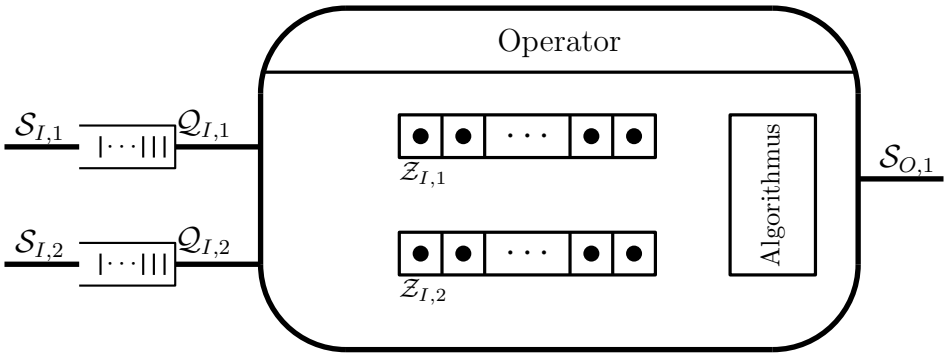


Abbildung 2.3: Aufbau eines Operators (abstrahiert)

elemente umgewandelt und mit dem aktuellen Zeitstempel versehen. Schließlich sendet die Datenquelle den neuen Datenstromwert innerhalb des DSMS an alle verbundenen Systemelemente.

Die Verarbeitung der Daten findet in Operatoren statt. Operatoren können einen oder mehrere Datenströme als Eingangsgröße besitzen. Ein Operator empfängt einen Datenstrom von einer Datenquelle oder einem anderen Operator und verarbeitet die Daten entsprechend seinem Algorithmus. Operatoren berücksichtigen dabei entweder ausschließlich den aktuellen Datenstromwert oder eine Sequenz von Werten eines Stroms. Ein Beispiel für einen Datenstromoperator, der nur den aktuellen Wert verarbeitet ist ein Filter. Entsprechend seinem Prädikat leitet der Filter das empfangene Datenstromelement weiter oder nicht. Ein Operator, der auch historische Werte eines Datenstroms berücksichtigt, ist die Aggregation. Auf einer Menge aus dem aktuellen Wert und den historischen Werten wird eine Operation durchgeführt, z. B. die Berechnung des Minimums oder des Durchschnitts über alle Werte. Der Verbund (engl. „Join“) repräsentiert einen Operator, der Sequenzen von mehreren Datenströmen verarbeitet. Dabei werden vergangene Werte der Eingangsströme im Arbeitsspeicher gehalten, um eintreffende Datenstromwerte mit ihnen zu verknüpfen.

Abbildung 2.3 stellt einen abstrahierten Operator dar. Die Eingangsdatenströme ($\mathcal{S}_{I,1}$, $\mathcal{S}_{I,2}$) werden in eigenen Eingangswarteschlangen ($\mathcal{Q}_{I,1}$, $\mathcal{Q}_{I,2}$) gepuffert. Die Werte werden für die Verarbeitung aus den Warteschlangen entnommen und gemäß dem eingesetzten Algorithmus behandelt. Die Ergebnisse werden über den Ausgangsdatenstrom ($\mathcal{S}_{O,1}$) an nachfolgende Systemelemente übertragen. Optional werden vergangene Werte in den inneren Zuständen ($\mathcal{Z}_{I,1}$, $\mathcal{Z}_{I,2}$) gespeichert.

Nach der Verarbeitung wird das Ergebnis an alle direkt verbundenen, nachfolgenden Systemelemente – Operatoren oder Datensenken – weitergegeben. Durch Datensenken verlassen die Daten schließlich das DSMS. Eine Datensenke besitzt eine Verbindung zu einer externen Applikation, z. B. einer Datenbank, einer Steuerung oder einer betriebswirtschaftlichen Anwendung. Die empfangenen Datenströme werden über diese Verbindung an die angeschlossenen Systeme weitergeleitet.

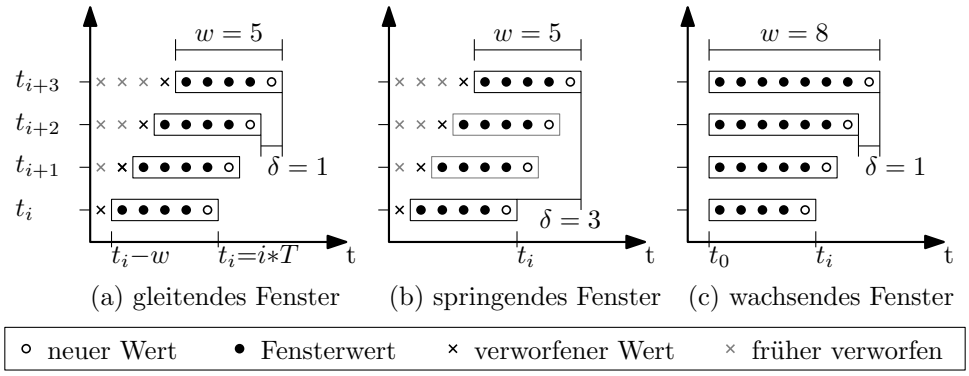


Abbildung 2.4: Auswahl von Fenstertypen – schematische Darstellung

2.2.4 Verwendung von Fenstern

Operatoren, die für ihre Verarbeitung vergangene Werte berücksichtigen, wirken sich blockierend auf die gesamte Verarbeitung im DSMS aus. Das bedeutet, ein Ergebnis kann erst dann produziert werden, wenn alle Datenstromwerte bekannt sind. Aufgrund der fortlaufenden Übermittlung von neuen Datenstromelementen ist diese Bedingung niemals erfüllt [BBD+02]. Eine Möglichkeit, blockierende Operatoren in nicht-blockierende Operatoren umzuwandeln, ist die Verwendung von sogenannten Fenstern (engl. „windows“). Mit Fenstern lässt sich die Menge der zu berücksichtigenden Datenstromwerte beschränken [CJ09]. Ein Fenster wird durch eine Fenstergröße w charakterisiert.

Für Fenster in Datenstromsystemen gibt es mehrere Unterscheidungskriterien [GÖ03a]. Eine erste Unterscheidung kann zwischen zeitbasierten und anzahlbasierten Fenstern getroffen werden. Ein zeitbasiertes Fenster[6] (auch „physisches Fenster“) speichert die Werte über einen festgelegten Zeitraum, z. B. alle Werte eines Temperatursensors der letzten 60 Minuten, d. h. $w = 60\,\text{min}$. Ein anzahlbasiertes Fenster (auch „logisches Fenster“) speichert hingegen eine Menge von Datenstromwerten, z. B. die letzten 1000 Werte eines Temperatursensors, d. h. $w = 1000$. Für periodische Datenströme ist die Festlegung der Fenstergröße als Zeit oder Anzahl äquivalent. Eine Zeit lässt sich unter Berücksichtigung des Eingangsverhaltens direkt in eine Anzahl überführen und umgekehrt.

Ein weiteres Kriterium zur Unterscheidung ist die Dynamik des Fensters. Die gebräuchlichste Form ist ein gleitendes Fenster (engl. „sliding window“). Bei anzahlbasierten Fenstern wird mit jedem neuen Datenstromwert der älteste Wert im Fenster verworfen (siehe auch Abbildung 2.4 (a)). Bei zeitbasierten Fenstern wird der Zeitstempel des neuen Wertes ts_{neu} und die Fenstergröße für das Verwerfen von alten Werten herangezogen. Ist der Zeitstempel eines Wertes im Fenster kleiner als $ts_{neu} - w$ wird dieser Wert verworfen. In aperiodischen Datenströmen mit gleitenden, zeitbasierten Fenstern

[6] Stellenweise wird in dieser Arbeit das Symbol w^* für zeitbasierte Fenster verwendet. Ist der Fenstertyp eindeutig, z. B. aufgrund der Einheit, oder nicht relevant, wird darauf verzichtet und wie in anderen Arbeiten w benutzt. Für anzahlbasierte Fenster wird immer w verwendet.

können somit auch mehrere Werte auf einmal verworfen werden.

Nach dem Aktualisieren des Fensters werden die darin befindlichen Daten verarbeitet und ein Ergebnis wird ausgegeben. Ein Operator mit einem gleitenden Fenster produziert also für jeden Eingangswert umgehend einen oder mehrere Ausgangswerte. Dieses Verhalten wird durch die Schrittweite δ ($\delta \in \mathbb{Z}, \delta \neq 0$) eines Fensters beschrieben. Bei gleitenden Fenstern ist $\delta = 1$. Für negative δ bewegt sich das Fenster rückwärts [GÖ03a]. Das erfordert auch, dass der Datenstrom rückwärts läuft, d. h. mit fallenden Zeitstempeln. Dies ist beispielsweise bei der Auswertung persistierter Daten möglich [CJ09]. Für $\delta > 1$ wird nicht mit jedem Eingangswert ein Ergebnis ausgegeben. Dieser Fenstertyp wird springendes Fenster genannt (engl. „jumping window“) und ist in Abbildung 2.4 (b) dargestellt. Bei ankommenden neuen Werten verhält sich dieses Fenster wie ein gleitendes Fenster. Ein alter Wert wird verworfen und der neue im Fenster gespeichert. Es wird jedoch nicht immer ein Ergebnis berechnet. Dies entspricht den grauen Fenstern in Abbildung 2.4 (b). Ein Fenster bei dem $\delta > w$ ist, wird von [CÇC$^+$02] als „tumbling window“ bezeichnet. In diesem Fall sind aufeinanderfolgende Fenster nicht überlappend, sie haben keine Datenstromwerte gemeinsam. Ein weiterer Fenstertyp ist das wachsende Fenster (engl. „landmark window“). Der Unterschied zu anderen Fenstern ist, dass keine Werte verworfen werden – ein Ende des Fensters ist fixiert. In Abbildung 2.4 (c) ist dieser Fenstertyp dargestellt. Das linke Ende des Fensters steht fest bei t_0. Die Fenstergröße wird mit jedem neuen Datenstromelement vergrößert.

In der Literatur werden weitere, spezialisierte Fenstertypen vorgestellt, beispielsweise Fenster mit unterschiedlicher Zeitgranularität (engl. „tilted time frame model“ [HK05]). Die Betrachtungen in dieser Arbeit beziehen sich in erster Linie auf die zuvor genannten, gewöhnlichen Fenstertypen, insbesondere gleitende Fenster, und nicht auf die Spezialfälle, wenngleich die Möglichkeit besteht, die vorgestellten Konzepte auch auf andere Fenstertypen anzuwenden.

Um unterschiedliche DSMS in Bezug auf die Semantik ihrer Anfrageausführung vergleichen zu können, wurde von [BDD$^+$10] ein Analysemodell vorgestellt, welches in erster Linie für Anfragen mit zeitbasierten Fenstern angewendet werden kann. Das sogenannte Secret-Modell nutzt vier Kategorien, um ein System zu beschreiben: (1) Scope, (2) Content, (3) Report und (4) Tick. Die ersten zwei Kategorien beziehen sich auf Anfragen bzw. die betrachteten Eingangsdaten. Die anderen zwei Kategorien beschreiben Systemeigenschaften. Die in dieser Arbeit verwendete Ausführungssemantik lässt sich im Sinne des Secret-Modells einer tupel-getrieben Verarbeitung zuordnen (Kategorie Tick). Dabei werden Fenster verarbeitet, wenn sich der Inhalt ändert (R_{cc}) und das Fenster nicht leer ist (R_{ne}) (Kategorie Report). Die im System verwendeten Zeitstempel sind systemzeitbasiert. Dieser Fall wurde in [BDD$^+$10] (Anhang B.1.1) diskutiert und ausgeschlossen, die Kategorie Content ist deshalb nicht klassifizierbar. Die Beschreibung der Kategorie Scope ist abhängig vom Zeitstempel des neuesten Datenstromelements ($\mathcal{D}.ts$) und der Fenstergröße, d. h. für die Kategorie Scope gilt bei zeitbasierten Fenstern $[\mathcal{D}.ts - w, \mathcal{D}.ts]$. Eine Erweiterung von Secret für anzahlbasierte Fenster wurde ebenfalls in [BDD$^+$10] (Anhang B.2.1) skizziert.

2.2.5 Gemeinsame Warteschlangen

Um den Speicherbedarf eines DSMS zu minimieren, kann bei Operatoren, welche Eingangsdaten von gleichen Vorgängern beziehen, das Prinzip der *gemeinsamen Warteschlange* (engl. „queue sharing“) angewendet werden. Demnach existiert eine Warteschlange, die alle direkt nachfolgenden Operatoren mit Eingangswerten beliefert. Um die Übersicht zu behalten, welche Werte weiterhin gespeichert bleiben und welche aus der Warteschlange entfernt werden können, müssen der Warteschlange alle nachfolgenden Operatoren und deren aktuelle Ausleseposition bekannt sein. Zudem können von Operatoren verschiedene Ausleserichtungen oder Auswahlmethoden benutzt werden. Die am häufigsten genutzte Methode ist das Auslesen in zeitlich ansteigender Reihenfolge basierend auf dem Zeitstempel der gespeicherten Werte, aber auch gezieltes Auslesen nach Attributwerten ist möglich. Wenn ein Datentupel von allen Operatoren ausgelesen wurde, kann es aus der Warteschlange entfernt werden.

In [MWA+03] wurde die Implementierung für das STREAM-System [ABB+04] beschrieben. Für die gemeinsame Nutzung einer Warteschlange wird pro nutzenden Operator ein Zeiger auf das nächste auszulesende Datentupel gespeichert. Die Verwaltung erfolgt durch einen Speichermanager. Datenwerte werden verworfen, wenn sie älter als das älteste referenzierte Tupel sind. Das Prinzip der gemeinsamen Warteschlange wird in [MWA+03] auch für innere Zustände diskutiert. Bei *gemeinsamen Zuständen* (synopsis sharing) sind die Herausforderungen aber deutlich komplexer. So werden die Fragen gestellt, welcher Operator für die Verwaltung eines gemeinsamen Zustands verantwortlich ist, oder wie mit unterschiedlichen Fenstergrößen, Fenstertypen oder Zugriffsraten umzugehen ist.

Eine weitere Implementierungsmöglichkeit einer gemeinsamen Warteschlange mit Auslesen in zeitlicher Reihenfolge wurde in [CÇC+02] für das System Aurora gezeigt. Für eine Warteschlange, die durch einen Speichermanager verwaltet wird, werden für jeden nachfolgenden Operator zwei Zeiger auf Datenelemente gespeichert. Dabei zeigt ein Zeiger auf die aktuelle Ausleseposition und der andere auf das älteste berücksichtigte Tupel, d. h. Ausleseposition abzüglich Fenstergröße. Alle Werte der Warteschlange, die älter sind als das älteste referenzierte Datentupel werden durch den Speichermanager verworfen.

Gemeinsame Warteschlangen und gemeinsame Zustände sind ein gutes Mittel, um den Speicherplatz in einem Datenstromsystem zu minimieren. Das Prinzip lässt sich jedoch ausschließlich in zentralisierten Systemen anwenden, da die Datenwerte im Hauptspeicher gehalten werden. Für verteilte Systeme ist diese Lösung somit nicht anwendbar. Beim Einsatz von geteilten Warteschlangen bzw. Zuständen ist zudem ein Verwaltungsaufwand zu berücksichtigen.

2.3 Zustandsbehaftete Operatoren

Operatoren, die ausschließlich den aktuellen (zuletzt empfangenen) Datenstromwert verarbeiten, werden als zustandslos bezeichnet. Beispiele sind einfache Filteroperatoren, die den Eingangswert mit einem Grenzwert vergleichen und den Eingangswert entsprechend weiterleiten oder nicht. Zustandsbehaftete Operatoren behalten auch vergangene Werte im Arbeitsspeicher und berücksichtigen diese für die Berechnung, z. B. die Werte im zugehörigen Fenster, das letzte Ergebnis oder ein Zwischenergebnis der letzten Berechnung. Werden zustandsbehaftete Operatoren in Betrieb genommen, füllen sich deren Zustände allmählich mit Werten. Während gespeicherte Ergebnisse oder Zwischenergebnisse[7] bereits nach einer Periode verfügbar sind, ist für viele Fenster eine längere Zeit einzuplanen. Die Zeitdauer, bis ein Fenster vollständig gefüllt ist, wird als Anlaufzeit (T_A) bezeichnet. Die nachfolgenden Betrachtungen beziehen sich hauptsächlich auf gleitende Fenster. Die Übertragbarkeit auf andere Fenstertypen ist gemäß den Eigenschaften der Fenster zu bewerten. Beispielsweise existiert für wachsende Fenster keine Anlaufzeit.

Befindet sich ein Fenster im „stationären Zustand“, d. h. es ist vollständig gefüllt, dann kann ein exaktes Ergebnis über dieses Fenster berechnet werden. Während der Anlaufzeit ist dies nicht möglich, da im Fenster Datenstromwerte fehlen. Somit können während der Anlaufzeit nur Näherungen über den projektierten Zeitraum oder die spezifizierte Anzahl errechnet werden.

Nachfolgend werden die Eigenschaften von zustandsbehafteten Operatoren mit einem Zustand und mehreren Zuständen sowie von Anfragen, die zustandsbehaftete Operatoren enthalten, betrachtet.

Operatoren mit einem Zustand Zur Gruppe der zustandsbehafteten Operatoren mit einem inneren Zustand gehören die Aggregationsoperatoren, welche Operationen wie Minimum, Maximum, gleitende Summe oder gleitender Mittelwert über ein Fenster von Datenstromwerten ausführen. Der allgemeine Aufbau ist in Abbildung 2.5 beispielhaft dargestellt. Ein Operator A errechnet für einen Datenstrom $\mathcal{S}_A$ eine gleitende Summe und gibt das Ergebnis als Datenstrom $\mathcal{S}'_A$ aus. Das eingestellte Fenster wird im Zustand $\mathcal{Z}_A$ gespeichert. Wird ein neuer Wert aus der Warteschlange $\mathcal{Q}_A$ entnommen, wird er in den Zustand eingefügt und alle Werte, die entsprechend der Fenstergröße zu alt oder zu viel sind, werden daraus verworfen. Mit Hilfe des referenzierten Algorithmus wird das Ergebnis ermittelt.

Einfache Implementierungen analysieren für jeden neuen Eingangswert das gesamte Fenster, um ein Ergebnis zu berechnen. Das hat einerseits den Vorteil, dass die Verarbeitung im Operator unabhängig vom Algorithmus ist. Andererseits wird dadurch unnötig viel CPU-Zeit verbraucht. Mit fortgeschrittenen Implementierungen kann der CPU-Aufwand reduziert werden [Mut05]. Die Abarbeitung im Operator kann sich aber für verschiedene Algorithmen unterscheiden, da gegebenenfalls unterschiedliche Arbeits-

[7] Sofern sie nicht von vollständigen Fenstern abhängig sind.

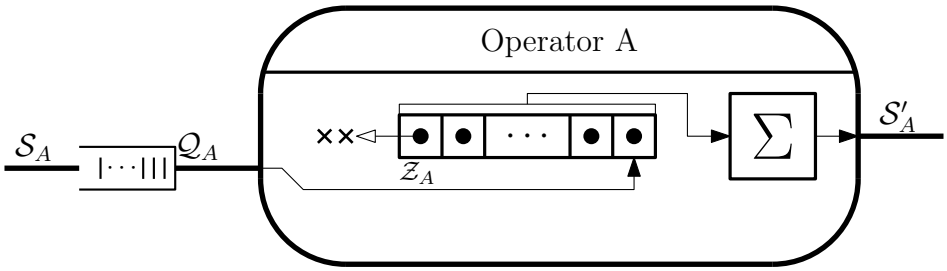


Abbildung 2.5: Aufbau eines Aggregationsoperators (Summe)

schritte durchzuführen sind oder verschiedene Zwischenergebnisse gespeichert werden.

Beispielsweise kann für die Berechnung einer gleitenden Summe das Ergebnis fortlaufend durch Addieren der neuen Werte und Subtrahieren der verworfenen Werte gebildet werden. Dadurch muss das Fenster nicht für jeden Wert vollständig ausgewertet werden, was CPU-Zeit spart. Zusätzlich zum Fenster ist das letzte Ergebnis im inneren Zustand des Operators zu speichern. Ist ein gleitender Mittelwert zu berechnen, ist die fortlaufende Summe abschließend durch die Anzahl der Werte im Fenster zu dividieren. Je nach Fenstertyp oder Implementierung ist unter Umständen die Anzahl als fortlaufende Größe mitzuführen. Für Operatoren mit Minimum- oder Maximum-Algorithmus sind ebenso Optimierungen möglich. Auch dafür ist gegebenenfalls das Speichern von Zwischenwerten nötig.

Die Anlaufzeit von Operatoren mit einem gleitenden Fenster ist abhängig von den Fenstereigenschaften und von den Signaleigenschaften des zugehörigen Eingangsdatenstroms. Die nachfolgenden Gleichungen basieren auf der Annahme, dass mit Eingangswerten das Fenster aktualisiert wird und alte oder überschüssige Werte verworfen werden.

Für zeitbasierte Fenster entspricht die Anlaufzeit der Fenstergröße $T_A = w^*$, unabhängig davon, ob es sich um einen periodischen oder aperiodischen Eingangsdatenstrom handelt. Einzige Bedingung ist, dass die (theoretische) Periodendauer oder der mittlere Ereignisabstand des Eingangsdatenstroms im Vergleich zur Fenstergröße hinreichend klein ist, d. h. $T < w^*$ bzw. $\overline{T} \ll w^*$. Anderenfalls ist T_A nicht von w^* abhängig. Als ungünstiges Beispiel kann man sich ein zeitbasiertes, gleitendes Fenster mit $w^* = 100\,\mathrm{s}$ vorstellen, angewendet auf einen Datenstrom mit $T = 10\,\mathrm{min}$. Das Fenster kann nur den aktuellen Wert enthalten und die Anlaufzeit beträgt im schlechtesten Fall 10 min. Vielmehr ist die Anlaufzeit für diese Konfiguration, wenn überhaupt, nur zur Laufzeit bestimmbar, da T_A der Zeit bis zum Eintreffen des nächsten Wertes entspricht. Da eine solche Konfiguration wenig sinnvoll erscheint, werden derartige Fälle für diese Arbeit ausgeschlossen. Dies wird auch für aperiodische Datenströme angenommen, wenngleich die Möglichkeit besteht, dass die Zeit bis zur Ankunft des nächsten Wertes die Fenstergröße w^* übersteigt. Für zeitbasierte Fenster wird also $T < w^*$ vorausgesetzt, und es gilt für die Anlaufzeit

$$T_A = w^*. \tag{2.3}$$

Bei anzahlbasierten Fenstern ist die Anlaufzeit lediglich für periodische und schwankungsbeschränkte Datenströme berechenbar, da diese deterministisch sind. Für die Berechnung der Anlaufzeit in einem periodischen Datenstrom sind die Periodendauer des Eingangsdatenstroms (T) sowie die Anzahl der eingehenden Werte pro Periode (N_i) und die Verarbeitungszeit eines Wertes (T_p) zu berücksichtigen. Die Anlaufzeit kann dann mit Hilfe der Abrundungsfunktion und der Modulofunktion gemäß Gleichung (2.4) berechnet werden. Für $T \gg T_p$ kann die Gleichung durch Vernachlässigung des zweiten Summanden vereinfacht werden.

$$T_A = \left\lfloor \frac{w}{N_i} \right\rfloor * T + (w \mod N_i) * T_p \tag{2.4}$$

Die Berechnung für schwankungsbeschränkte Datenströme muss zusätzlich die Schwankungen τ und τ' berücksichtigen. Zudem ist T für diesen Datenstromtyp als theoretische Periodendauer definiert. Deshalb kann für T_A lediglich ein Intervall angegeben werden. Errechnet man T_A^* mit Hilfe von Gleichung (2.4), dann ergibt sich für die Anlaufzeit in schwankungsbeschränkten Datenströmen

$$T_A^* - \tau - \tau' \leq T_A \leq T_A^* + \tau + \tau'. \tag{2.5}$$

Werden gleitende Fenster auf aperiodische Datenströme angewendet, kann die Anlaufzeit nur abgeschätzt werden, da das Verhalten des Eingangsdatenstroms nicht vorhersagbar ist. Unter Verwendung des mittleren Ereignisabstands $\overline{T}$ lautet die geschätzte Anlaufzeit

$$T_A \approx \frac{w}{N_i} * \overline{T}. \tag{2.6}$$

Sicherlich ist es möglich, mit umfassenderen Modellen die Abschätzung von T_A zu präzisieren, insbesondere für aperiodische Datenströme. Dies ist jedoch nicht Schwerpunkt dieser Arbeit, weshalb zur Erläuterung der Migrationsprinzipien diese vereinfachten Modelle verwendet werden.

Operatoren mit mehreren Zuständen Operatoren können zwei oder mehr Eingangsströme und dementsprechend auch innere Zustände besitzen. Zu diesen Operatoren gehören die Verbundoperatoren, von denen es zahlreiche Ausprägungen mit zwei oder mehreren Eingangsdatenströmen (engl. „binary join“ bzw. „n-ary join“, „multi-way join“ oder „MJoin“ [GÖ03b, VNB03, DIR07]) gibt, wobei sich Verbundoperatoren mit mehreren Eingängen auch aus jenen mit zwei Eingängen modellieren lassen[8].

[8] Bei Kompositionen von Verbundoperatoren in Datenströmen ist die Gewährleistung einer semantischen Äquivalenz deutlich schwieriger als in relationalen Datenbanken.

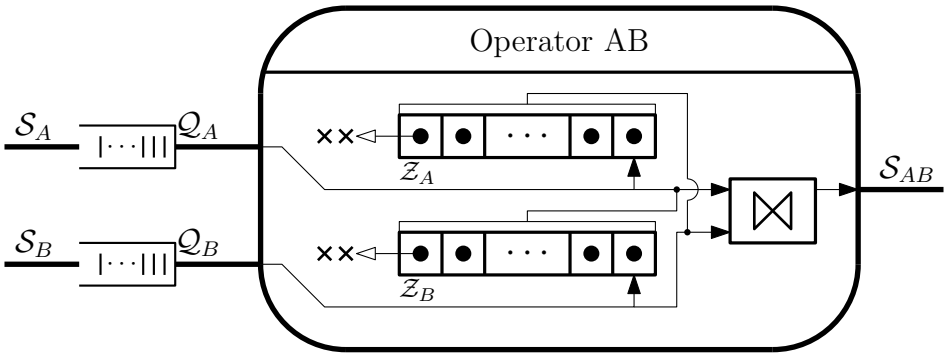


Abbildung 2.6: Aufbau eines Verbundoperators (Binary Join)

In Abbildung 2.6 ist der Aufbau eines Operators mit mehreren inneren Zuständen am Beispiel eines Verbundoperators dargestellt. Dieser verbindet mittels Verbundalgorithmus Datenstromwerte der Eingangsdatenströme $\mathcal{S}_A$ und $\mathcal{S}_B$ miteinander und gibt die Ergebnistupel als Datenstrom $\mathcal{S}_{AB}$ aus[9]. Die Verbundoperation wird durch den Verbundalgorithmus realisiert und entspricht einem kartesischen Produkt zweier Eingangsgrößen mit einer anschließenden Selektion, welche gewöhnlich auf einem Vergleich (Verbundprädikat) von Attributwerten basiert.

Die Verarbeitung innerhalb des Operators führt, am Beispiel von Eingangsstrom $\mathcal{S}_A$, die folgenden Schritte durch: Nach der Entnahme eines Eingangswertes aus der Warteschlangen $\mathcal{Q}_A$, wird der Zustand des anderen Eingangsstroms aktualisiert, d. h. in $\mathcal{Z}_B$ werden Werte, welche außerhalb des Fensters liegen, verworfen. Anschließend wird der Eingangswert gemäß dem Verbundprädikat mit den Werten des aktualisierten Zustands zusammengeführt und als Ergebnis ausgegeben. Sofern die Ergebnisse nicht gefiltert werden (Selektivität $sel = 1$), entspricht die Anzahl der Ergebnisse zu einem Eingangswert der Anzahl der Werte im aktualisierten Zustand. Abschließend wird der Eingangswert im eigenen Zustand gespeichert, d. h. in $\mathcal{Z}_A$. Die Verarbeitung kann wiederholt werden, bis die Warteschlange leer ist. Gleichermaßen wird für Werte des Eingangsstroms $\mathcal{S}_B$ der Zustand $\mathcal{Z}_A$ aktualisiert und für die Ergebnisbildung genutzt, bevor der Eingangswert schließlich in $\mathcal{Z}_B$ abgelegt wird.

Beim Zusammenfügen der Tupel im Verbundoperator wird das Ergebnistupel mit einem Zeitstempel versehen. Hier kommen die zuvor beschriebenen zusammengesetzten Zeitstempel zum Tragen. Der Zeitstempel des Ergebnistupels ist die Menge aller Zeitstempel der Sub-Tupel.

Maßgeblich für das Verhalten von Verbundoperatoren, insbesondere in mehrstufigen Anfragen („Verbundbäume“), ist die Art und Weise, wie Datentupel aus den Zuständen verworfen werden. In Systemen, die vergleichbare Migrationsstrategien[10] vorstellen

[9] Es wird in dieser Arbeit die gleiche Vorgehensweise und die gleiche Semantik wie im CAPE-System [ZRH04] genutzt.

[10] Vergleichbare Systeme werden im nächsten Kapitel im Detail diskutiert.

[Zhu06, YKPS07, Krä07], werden alle Werte aus den Zuständen verworfen, die mindestens ein Sub-Tupel enthalten, welches außerhalb des zeitbasierten Fensters gültig war. Anders ausgedrückt, es müssen alle Basistupel innerhalb des Fensters generiert worden sein, um ein Ergebnistupel zu erzeugen. Aus diesem Grund wird ts_{min} für das Verwerfen benötigt. Ein Zustandstupel mit dem Zeitstempel ts_{state} wird auf Basis des Zeitstempels des Eingangswertes (ts_{input}) aus dem Zustand verworfen, wenn $max(ts_{input}) - min(ts_{state}) > w$. Durch diese Methode können Anfragen durch Umsortieren der Operatoren optimiert werden und bleiben gleichzeitig semantisch äquivalent. Sofern nicht anders angegeben, wird die Methode auch in dieser Arbeit genutzt. Die entwickelten Migrationskonzepte sind aber prinzipiell unabhängig davon und deshalb auch für andere Ansätze verwendbar.

In Systemen, die keine zusammengesetzten, sondern nur einfache Zeitstempel unterstützen, hat die Zeitstempelvergabe unterschiedliche Schwierigkeiten zur Folge. Es werden im Folgenden zwei Fälle kurz diskutiert. Entspricht der Zeitstempel des Ergebniselements dem kleinsten Zeitstempel der Eingangswerte, also $\mathcal{D}_{AB}.ts = \min(\mathcal{D}_A.ts, \mathcal{D}_B.ts)$, wie beispielsweise im System Aurora [ACÇ+03], dann verbleiben bei Verbundbäumen mehr Tupel in den inneren Zuständen als mit dem oben beschriebenen Ansatz. Es werden also auch Ergebnistupel ausgegeben, deren Sub-Tupel weiter als w auseinander liegen. Der gleiche Effekt ist zu beobachten, wird ein neuer, aktueller Zeitstempel für ein Ergebnis erzeugt oder der Maximalwert der Sub-Tupel genutzt. Dadurch gehen die Informationen über das Alter der älteren Sub-Tupel verloren, weshalb ein Verbundbaum mit k Ebenen auch Ergebnisse produziert, die Sub-Tupel enthalten können, welche so alt sind, wie $k * w$ groß ist, die Verwendung einer globalen Fenstergröße vorausgesetzt. Die Ergebnismenge hängt somit von der Position der einzelnen Operatoren in der Anfrage ab. Egal, ob die Minimal- oder Maximalmethode für den Ergebniszeitstempel genutzt wird, es ist nicht möglich, durch Umsortieren der Verbundoperatoren eine semantisch äquivalente Anfrage zu erhalten. Schwierigkeiten infolge der Zeitstempelvergabe, vor allem im Zusammenhang mit Operatoren mit zwei Eingängen, wurden auch in [BBD+02] diskutiert.

Für die Bestimmung der Anlaufzeit von Operatoren mit mehreren Zuständen kann auf die Erkenntnisse der einfachen, zustandsbehafteten Operatoren zurückgegriffen werden. Es ist jedoch der Umgang der Operatoren mit den Zuständen zu berücksichtigen, hauptsächlich das Verwerfen und die Verwendung der Zustände zur Ergebnisberechnung.

Daraus leiten sich zwei Überlegungen ab. Erstens, innere Zustände werden unter Umständen nicht nur von ihren eigenen Eingangsdatenströmen beeinflusst. Am Beispiel der Verbundoperatoren wurde bereits gezeigt, dass das Verwerfen aus den Zuständen durch den gegenüberliegenden Eingangsdatenstrom ausgelöst wird, z. B. bei $\mathcal{Z}_A$ durch Eingangswerte von $\mathcal{S}_B$. Aus diesem Grund kann die Anlaufzeit in periodischen Datenströmen bis zu eine Periodendauer länger sein. In schwankungsbeschränkten Datenströmen kommt außerdem die Schwankung hinzu. In aperiodischen Datenströmen kann keine genaue Angabe gemacht werden. Gleichung (2.7) repräsentiert eine allgemeine Abschätzung für die Anlaufzeit eines Zustands unter Verwendung der Größen T'_A und $T^{\#}$, welche

entsprechend den Eigenschaften des jeweiligen Eingangsstroms eines Zustandes ermittelt werden. Für T'_A ist demgemäß eine der Gleichungen (2.3) bis (2.6) zu benutzen. Die Größe $T^{\#}$ steht für die zusätzliche Zeit je nach Datenstromtyp wie oben beschrieben. In aperiodischen Datenströmen ist ein geeigneter Wert für c zu wählen.

$$T'_A \leq T_A \leq T'_A + T^{\#} \tag{2.7}$$

$$\text{mit } T^{\#} = \begin{cases} T & \text{periodischer Datenstrom} \\ T + \tau + \tau' & \text{schwankungsbeschränkter Datenstrom} \\ c * \overline{T} & \text{aperiodischer Datenstrom} \end{cases}$$

Zweitens, für jeden inneren Zustand eines Operators kann eine Anlaufzeit bestimmt werden. Die Anlaufzeit eines Operators ist die größte dieser Anlaufzeiten. Für einen Operator mit k inneren Zuständen entspricht die Anlaufzeit der Gleichung (2.8), wobei jedes $T_{A,i}$ gemäß Gleichung (2.7) ermittelt wird.

$$T_A = \max(\{T_{A,1}, \ldots, T_{A,i}, \ldots, T_{A,k}\}), \quad \forall i, k \in \mathbb{N}^+ \text{ und } i \leq k \tag{2.8}$$

Die Bestimmung der Anlaufzeit für zustandsbehaftete Operatoren mit mehreren inneren Zuständen ist davon abhängig, wofür T_A später verwendet werden soll. Geht es ausschließlich um die Abschätzung der Anlaufzeit, um festzustellen, ab wann ein Operator definitiv exakte Ergebnisse liefert, ist der ungünstigste Wert zu ermitteln (obere Grenze in Gleichung (2.7)). Ist ein Vergleich zur Migrationsdauer gesucht, ist die untere Grenze interessanter.

Anfragen Anlaufzeiten lassen sich nicht nur für einzelne Operatoren sondern auch für Anfragen oder Teilanfragen bestimmen, die zustandsbehaftete Operatoren enthalten. Grundsätzlich gilt, die Anlaufzeit einer Anfrage entspricht der maximalen Anlaufzeit aller enthaltenen Operatoren. Es kann also die Gleichung (2.8) analog für eine Anfrage genutzt werden, setzt man wiederum die Anlaufzeiten der Operatoren dieser Anfrage ein.

Für die Bestimmung der Anlaufzeiten der einzelnen Operatoren sind jeweils die Kennwerte ihrer Fenster und Eingangsdatenströme zu verwenden. Unter Eingangsdatenstrom ist dabei entweder ein Eingangsdatenstrom der Anfrage oder der Ausgangsdatenstrom des Vorgängeroperators zu verstehen, je nachdem, an welcher Position sich der zu bewertende Operator in der Anfrage befindet. So kann es auch für Anfragen mit periodischen Eingangsdatenströmen zu nichtdeterministischem Verhalten innerhalb der Anfrage kommen. Bezieht beispielsweise ein Operator mit einem anzahlbasierten Fenster seine Eingangsdaten von einem Filteroperator, so handelt es sich um einen aperiodischen Eingangsdatenstrom, und es ist mit den bereits beschriebenen Eigenschaften zu rechnen.

Zusammenfassung Bei der Inbetriebnahme von zustandsbehafteten Operatoren und Anfragen treten immer Anlaufzeiten auf, so auch bei der Migration, die genutzt wird,

um Operatoren oder Anfragen anzupassen oder auf andere Bearbeitungsknoten zu verlagern. In ungünstigen Fällen können lange Anlaufzeiten auftreten, wenn entsprechende Operatoren in den zu migrierenden Anfragen enthalten sind. Mit der Zielsetzung, exakte Ergebnisse zu erhalten, ist sicherzustellen, dass die Fenster schnellstmöglich gefüllt werden, um die Migration so schnell wie möglich abzuschließen – im Idealfall schneller als es mit Datenstromwerten der Eingangsdatenströme möglich ist.

2.4 Anpassen von Datenstromanfragen

Das Anpassen von Datenstromanfragen wurde in der Motivation als Notwendigkeit identifiziert, aber auch einige Herausforderungen sind bei der Durchführung von Anpassungen zu berücksichtigen. Hervorzuheben sind Echtzeitausgabe und Zustandserhaltung.

Ausgangspunkt einer Anpassung ist eine existierende Anfrage oder Teilanfrage, die „Originalanfrage“ oder „alte Anfrage“. Durch manuelle oder automatische Modifikation, meistens mit dem Ziel einer Optimierung, entsteht eine veränderte Anfrage oder Teilanfrage, auch „Zielanfrage“ oder „neue Anfrage“. Der Anpassungsprozess wird Migration genannt, und es wird die alte Anfrage in die optimierte, neue Anfrage überführt. Die Migration entspricht der Anlaufphase der neuen Anfrage. Dieser Prozess nimmt Zeit in Anspruch, die Migrationsdauer T_M. Sie erstreckt sich vom Migrationsbeginn ($t_{M,start}$) bis zum Migrationsende ($t_{M,end}$). Aus den Ursachen für eine Anpassung, z. B. wechselnde Bedingungen, neue Sensorik, Testen und Nutzen effizienterer Algorithmen oder Optimierung der Verarbeitungsreihenfolge, resultieren verschiedene Anpassungsmöglichkeiten, d. h. veränderte Parameter, neue Algorithmen oder eine modifizierte Anfragenkomposition.

Im Zusammenhang mit Original- und Zielanfrage wird häufig eine semantische Äquivalenz erwähnt [Zhu06, YKPS07, Krä07]. Formale Definitionen für die semantische Äquivalenz von Datenströmen und Anfragen wurden von [Krä07] beschrieben. Demnach sind Anfragen genau dann semantisch äquivalent, wenn ihre Ausgangsdatenströme jederzeit die gleichen Ergebnisse enthalten, setzt man die gleichen Eingangsdatenströme voraus. Existierende Migrationsverfahren erfordern eine semantische Äquivalenz der Original- und Zielanfrage. Inwiefern sich die Definition der semantischen Äquivalenz auch auf innere Zustände übertragen lässt, bleibt zu untersuchen. Zwar stellen innere Zustände vergangene Datenstromwerte dar und eine mögliche Definition könnte von der Äquivalenzdefinition von Datenströmen abgeleitet werden, es müssen aber zusätzlich verschiedene Eigenschaften, wie Fenstertyp und Fenstergröße, berücksichtigt werden.

Anpassungen von Anfragen, die zustandsbehaftete Operatoren enthalten, weisen neben den bereits genannten, allgemeinen Herausforderungen noch besondere Eigenschaften auf. Diese werden nachfolgend am Beispiel der Veränderung der Fenstergröße erläutert. Ausgangspunkt ist eine kontinuierlich arbeitende Originalanfrage mit mindestens einem Operator mit einem vollständig gefüllten, gleitenden Fenster. Ziel ist die Modifi-

kation der Fenstergröße[11]. Es ist davon auszugehen, dass Original- und Zielanfrage mit vollständig initialisierten Zuständen exakte Ergebnisse produzieren. Bei der Originalanfrage ist dies zu Migrationsbeginn der Fall. Bei der modifizierten Anfrage, die mit einem leeren Fenster beginnt, ist durch Migration dieser Status herzustellen, gegebenenfalls unterstützt durch eine Zustandserhaltung. Ist ein vollständiger innerer Zustand des Operators in der neuen Anfrage erreicht, endet die Migration. Bis dahin befindet sich der Operator in der Anlaufphase und sein Fenster ist nur teilweise gefüllt. Es können in den meisten Fällen[12] keine exakten Ergebnisse mit einem unvollständigen Fenster ermittelt werden. Es stellt sich die Frage, wie Ergebnisse von Anfragen zu bewerten sind, die unvollständige Zustände enthalten. Betrachtet man die Übergangszeit von der Originalanfrage zur vollständig initialisierten, modifizierten Anfrage, dann können folgende Fälle unterschieden werden.

Wird die Fenstergröße eines Operators verringert, sind weniger Datentupel für die Verarbeitung notwendig. Bei einer direkten Anpassung des Originaloperators können die überflüssigen Tupel sofort oder mit dem nächsten neuen Tupel verworfen werden. Die Bearbeitung kann dann lückenlos fortgesetzt werden. Wird jedoch ein neuer, modifizierter Operator benutzt, unter Umständen auf einem anderen Bearbeitungsknoten, dann ist dessen innerer Zustand neu aufzubauen. Ein Zustandstransfer hilft, den Zustand schneller zu füllen. Dabei ist es wichtig, den Transfer auf die notwendige Teilmenge des Originalzustands zu begrenzen, damit keine Datentupel übertragen werden, die nie zu einem Ergebnis beitragen. Das Gleiche gilt, wenn die Fenstergröße nicht verändert wird, beispielsweise wenn der Operator auf einen anderen Verarbeitungsknoten verschoben werden soll.

Beim Vergrößern der Fenstergröße sind mehr Datentupel nötig, als im Zustand des Originaloperators zu Migrationsbeginn vorhanden sind. Die fehlenden Werte müssen dann mit neuen Datenstromwerten aufgefüllt werden, was je nach Konfiguration und Systemeigenschaften auch längere Zeit in Anspruch nehmen kann.

Ein Beispiel für eine Fenstervergrößerung ist in Abbildung 2.7 dargestellt. Der Originaloperator mit einem gleitenden Fenster der Größe w_o verarbeitet einen Datenstrom $\mathcal{S}$ (Abbildung 2.7a). Ein neuer Operator mit der Fenstergröße $w_n > w_o$ kann nicht mit einem vollständig gefüllten Fenster beginnen, selbst wenn der Zustand des Originaloperators umgehend in den Zustand des neuen Operators eingefügt werden kann (Abbildung 2.7c). Da der Originaloperator bereits Werte des Eingangsdatenstroms aus seinem Zustand verworfen hat, weil sie für ihn nicht mehr relevant sind, besitzt der neue Operator ein unvollständiges Fenster. Er kann bezüglich der projektierten Fenstergröße nur genäherte Ergebnisse berechnen, weil ein Teil der Daten unbekannt ist. Im Verlauf der Migration erfasst der neue Operator in seinem Fenster alle Eingangswerte, ohne Werte aus dem Fenster zu verwerfen. Dieses Verhalten entspricht einem wachsenden Fenster.

[11] Original- und Zielanfrage sind dann nicht semantisch äquivalent.

[12] Unter bestimmten Umständen ist dies doch möglich. Es werden entsprechende Fälle im Zusammenhang mit den Zustandstransferstrategien beschrieben.

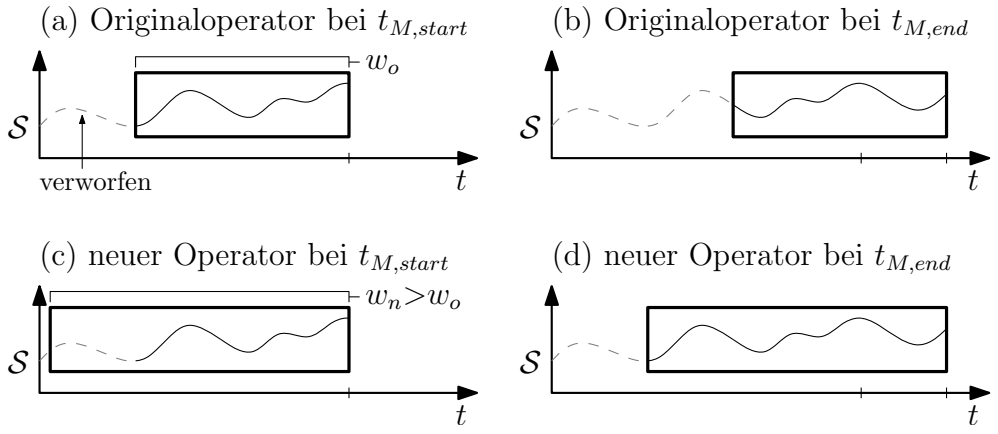


Abbildung 2.7: Fenstervergrößerung (qualitativ)

Ist die geplante Fenstergröße erreicht (Abbildung 2.7d), ist die Migration beendet und der neue Operator arbeitet mit einem gleitenden Fenster weiter. Ab diesem Zeitpunkt ist er in der Lage, exakte Ergebnisse zu ermitteln. Spätestens dann kann der neue Operator den Originaloperator in der Anfrage ersetzen. Der Originaloperator kann entfernt werden. Bereits während der Migration besteht die Möglichkeit, dass der neue Operator bessere Ergebnisse als der Originaloperator produziert, da er mehr Eingangsdaten berücksichtigt. Dies ist beispielsweise bei der Berechnung eines gleitenden Mittelwertes der Fall. Es können Ergebnisse mit steigender Genauigkeit berechnet werden bis die Zielkonfiguration erreicht ist, weshalb es sinnvoll sein kann, auch eher zum neuen Operator zu wechseln. Es liegt im Ermessen des Benutzers, einen früheren Wechsel zu erlauben. Außerdem ist festzustellen, dass die neue und die alte Anfrage semantisch nicht äquivalent sind. Das Gleiche gilt für die Zielanfrage während der Migration, da sich die aktuelle Fenstergröße fortlaufend ändert.

Durch die Änderung der Semantik kann es dazu kommen, dass nachfolgende Operatoren angepasst werden müssen. Eine solche „bedingte Änderung“ und vor allem deren Erkennung stellt eine besondere Herausforderung dar. Beeinflussende Größen sind u. a. die Charakteristik der Eingangsdatenströme, die Parameter und Algorithmen der Operatoren, die Fenstereigenschaften und die Komposition der Anfrage. Eine automatische Erkennung von Abhängigkeiten zwischen Operatoren in einer Anfrage sowie der Einfluss von Änderungen in Datenstromanfragen ist ein offenes Forschungsthema und wird hier nicht weiter betrachtet, da es für den Migrationsablauf unerheblich ist.

Das Anpassen von Datenstromanfragen während der Laufzeit ist eine wesentliche Grundlage für einen kontinuierlichen und lange laufenden Betrieb solcher Anfragen. Am Beispiel der Veränderung der Fenstergröße ist zu erkennen, dass neben Echtzeitausgabe und Zustandserhaltung auch andere Aspekte berücksichtigt werden müssen. So zeigt die Verkleinerung eines Fensters, dass nicht alle Werte für die Inbetriebnahme eines neuen Operators relevant sind. Stattdessen ist eine Auswahl der notwendigen Werte

aus dem Zustand des Originaloperators zu realisieren. Die Fenstervergrößerung demonstriert, dass die Anpassung Zeit in Anspruch nehmen kann, und dass bei der Migration ein Anlaufverhalten des neuen Operators zu beobachten ist. Außerdem gibt es mehr als eine Möglichkeit, wann zum neuen Operator umgeschaltet wird. All diese Aspekte beschränken sich nicht nur auf die Änderung von Fenstern, sondern sind generell bei der Konfiguration und Durchführung der Migration zu beachten. Die vorliegende Arbeit präsentiert Lösungen für diese und weitere Probleme bei der Anpassung von Datenstromanfragen während der Laufzeit. Existierende Ansätze werden im nachfolgenden Kapitel untersucht.

2.5 Anforderungen an eine zustandserhaltende Laufzeitadaption

Allgemeine Anforderungen an Datenstromsysteme in Echtzeitumgebungen wurden in [SÇZ05] beschrieben. Einige davon sind für den Anpassungsprozess von Datenstromanfragen zu berücksichtigen. Außerdem wurden am Motivationsbeispiel „Zustandsorientierte Wartung“ verschiedene anwendungsspezifische Anforderungen dargestellt. Nachfolgend werden alle migrationsrelevanten Anforderungen zusammengefasst und im Kontext der Migration erläutert.

- *Aktives Verarbeitungsmodell:* Daten werden fortlaufend als Strom verarbeitet, ohne sie vorher zu speichern. Für den Austausch innerhalb des Systems ist ein aktives Kommunikationsmodell zu verwenden, d. h. Push-basierte Datenübertragung und kein Polling[13] [SÇZ05]. Dies gilt auch für die Migration, insbesondere für den Zustandstransfer.

- *Verarbeitung und Ergebnisausgabe in Echtzeit:* Ein Datenstromsystem ist dahingehend zu optimieren, Ergebnisse trotz eines hohen Datenvolumens in Echtzeit zu produzieren und auszugeben [SÇZ05]. Damit verbunden ist eine garantierte Ergebnisausgabe. So darf auch während des Migrationsverlaufs der operative Betrieb, also die reguläre Verarbeitung der Daten, nicht verlangsamt oder gar ausgesetzt werden.

- *Verteiltes System:* Um eine hohe Skalierbarkeit zu erreichen, muss ein DSMS in einer verteilten Umgebung ausführbar sein [SÇZ05]. Das erfordert, dass Zustände bei Bedarf über das Netzwerk transferiert werden. Migrationsstrategien müssen diesem Umstand Rechnung tragen.

- *Transparente Migration:* Die Migration ist für den operativen Betrieb transparent auszuführen und darf ihn nicht negativ beeinflussen. Deshalb ist die Migration als

[13] Unter Polling versteht man das fortlaufende Abfragen von Daten durch den Empfänger. Polling führt im Vergleich zum Push-Modell zu einem erhöhten Kommunikationsaufwand.

Hintergrundprozess unter Verwendung der ungenutzten CPU-Zeit durchzuführen. In periodischen und schwankungsbeschränkten Datenströmen ist eine Kollision von Migrationsaktivitäten mit der regulären Verarbeitung auszuschließen. Da in aperiodischen Datenströmen eine Kollisionsvermeidung nicht möglich ist, ist die Verzögerung des operativen Betriebs, verursacht durch den Migrationsprozess, möglichst gering zu halten.

- *Zusätzlicher Speicher:* Für die Migration ist möglichst wenig zusätzlicher Speicher zu verwenden.
- *Exakte Ergebnisse:* Während der Migration sind exakte Ergebnisse Näherungen vorzuziehen. Ausnahme bilden Fälle, in denen Näherungen eine größere Genauigkeit im Vergleich zur Originalanfrage versprechen, z. B. bei Fenstervergrößerungen.
- *Kurze Migrationsdauer:* Neben der garantierten Ausgabe von Ergebnissen und ihrer Berechnung in Echtzeit ist eine Migration so schnell wie möglich zu realisieren. Deshalb ist der Zustandstransfer auf notwendige Werte zu beschränken.

Kapitel 3

Laufzeitadaption in Datenstromsystemen

Dieses Kapitel beschreibt Arbeiten, die direkt oder indirekt zur Thematik beitragen. Zunächst werden existierende Arbeiten analysiert, die Migrationsstrategien zur Laufzeitadaption in Datenstromsystemen vorgestellt haben. Weiterhin werden Datenstromsysteme betrachtet, die Methoden zur Laufzeitadaption oder zum Zustandstransfer besitzen, diese aber nicht in Verbindung mit Migrationsstrategien verwenden. In einer Übersicht werden alle betrachteten Migrationsstrategien vergleichend gegenübergestellt. Abschließend werden die wichtigsten Erkenntnisse zusammengefasst.

Konzepte zur Anpassung von laufenden Datenstromanfragen oder Teilanfragen wurden seit 2004 in der Literatur beschrieben. Ausgangspunkt sind eine existierende, zu optimierende Anfrage („Originalanfrage“) und eine neue, optimierte Anfrage[1] („Zielanfrage“). Originalanfrage und Zielanfrage können auch als Teilanfragen oder einzelne Operatoren vorliegen. Alle bekannten Migrationsverfahren beruhen auf zwei Grundprinzipien – Zustandstransfer oder Parallelmethode – oder einer Kombination beider. Beim Zustandstransfer wird der innere Zustand der Originalanfrage extrahiert und zur Initialisierung der Zielanfrage benutzt. Die Parallelmethode führt zeitweise Original- und Zielanfrage parallel aus und kommt ohne Zustandstransfer aus. Nachfolgend werden die existierenden Migrationslösungen vorgestellt und in Bezug auf die beschriebenen Anforderungen analysiert.

3.1 CAPE und D-CAPE

Die ersten Migrationsverfahren wurden für das System CAPE (Constraint-exploiting Adaptive Processing Engine [CAP09, RDZ+05]) beschrieben. Es wurden zwei Verfahren vorgestellt, die je eines der Grundprinzipien implementieren. Beide Verfahren wurden speziell für Bäume von Verbundoperatoren entwickelt. Original- und Zielanfrage werden mit „alter Plan“ bzw. „neuer Plan“ bezeichnet. Die für die Migration relevanten

[1] Das Finden der Zielanfrage wird von einigen Systemen unterstützt, steht in dieser Arbeit jedoch nicht im Mittelpunkt.

Operatoren, die eine gesamte Anfrage oder eine Teilanfrage darstellen können, werden als „Box“ zusammengefasst. Somit findet die Migration zwischen „alter Box“ und „neuer Box“ statt. Als Anforderung wird genannt, dass beide Boxen semantisch äquivalent sein müssen, d. h. beide Boxen haben die gleichen Ein- und Ausgänge und erzeugen für gleiche Eingangsdaten die gleichen Ergebnisse. Die Anfrageoptimierung erfolgt durch Umsortieren von Operatoren innerhalb einer Anfrage – eine Kompositionsänderung. Im Gegensatz zur Kompositionsänderung wurde eine Änderung von Parametern oder Algorithmen nicht beschrieben.

Ein Migrationsbeispiel aus [ZRH04] ist in Abbildung 3.1 dargestellt. Die Originalanfrage (Abbildung 3.1a) besteht aus drei Verbundoperatoren ($\bowtie$), welche nacheinander die vier Eingangsdatenströme ($\mathcal{S}_A$, $\mathcal{S}_B$, $\mathcal{S}_C$, $\mathcal{S}_D$) in einen Ausgangsdatenstrom $\mathcal{S}_{ABCD}$ umwandeln. Jeder Operator verarbeitet zwei Eingangsströme und verwaltet zu jedem Eingangsstrom ein gleitendes Fenster von Werten, z. B. $\mathcal{Z}_A$ für den Eingangsdatenstrom $\mathcal{S}_A$ oder den Zwischenzustand $\mathcal{Z}_{ABC}$ für $\mathcal{S}_{ABC}$. Der innere Zustand eines Verbundoperators setzt sich somit aus je einem gleitenden Fenster über die beiden Eingangsdatenströme zusammen. In [ZRH04] wird für alle Operatoren einer Anfrage eine *globale Fenstergröße* angenommen, d. h. alle Fenster aller Operatoren der Originalanfrage haben die gleiche Größe, z. B. $w = 180\,\mathrm{s}$.

Im Beispielszenario wurde die Zielanfrage (Abbildung 3.1b) als Optimierung der Originalanfrage identifiziert. Da die Migration alle Operatoren betrifft, sind Box und Plan im Beispiel identisch. Der neue Plan ist semantisch äquivalent zum alten, u. a. aufgrund der Verbundimplementierung mit der Methode zum Verwerfen. Auch im neuen Plan gilt eine globale Fenstergröße – die gleiche wie für die Originalanfrage. Nachfolgend werden die zwei Migrationsstrategien des CAPE-Systems vorgestellt.

3.1.1 Die Migrationsstrategie „Moving State“

Die Migrationsstrategie „Moving State“ (MS) [ZRH04, Zhu06] ist eine Umsetzung des Grundprinzips Zustandstransfer. Die Migration vom alten zum neuen Plan wird im Wesentlichen in drei Schritten durchgeführt. Mit dem Migrationsbeginn wird die Ausführung der Originalanfrage pausiert. Die Warteschlangen der Eingangsdatenströme sammeln während der Migration eingehende Datentupel. Diese werden aber nicht verarbeitet. Um Inkonsistenzen zu vermeiden, werden anfangs alle Datentupel verarbeitet, die sich in den Eingangs- und Zwischenwarteschlangen befinden und einen Zeitstempel besitzen, der vor dem Migrationsbeginn liegt ($ts < t_{M,start}$). Danach werden die Eingangs- und Ausgangswarteschlangen der alten und neuen Box miteinander verbunden. Anschließend beginnt die Zustandsmigration.

Zustandsvergleich („state matching“): In diesem Schritt werden alter und neuer Plan miteinander verglichen, um gleiche Zustände zu finden. Nur diese kommen für einen Zustandstransfer in Frage. Das sind in jedem Fall alle Eingangspuffer, da die Pläne semantisch äquivalent sind. Zudem können Zwischenzustände gleich sein. Im oberen Beispiel sind alle Zustände der Eingangsströme ($\mathcal{Z}_A$, $\mathcal{Z}_B$, $\mathcal{Z}_C$, $\mathcal{Z}_D$) in beiden Plänen

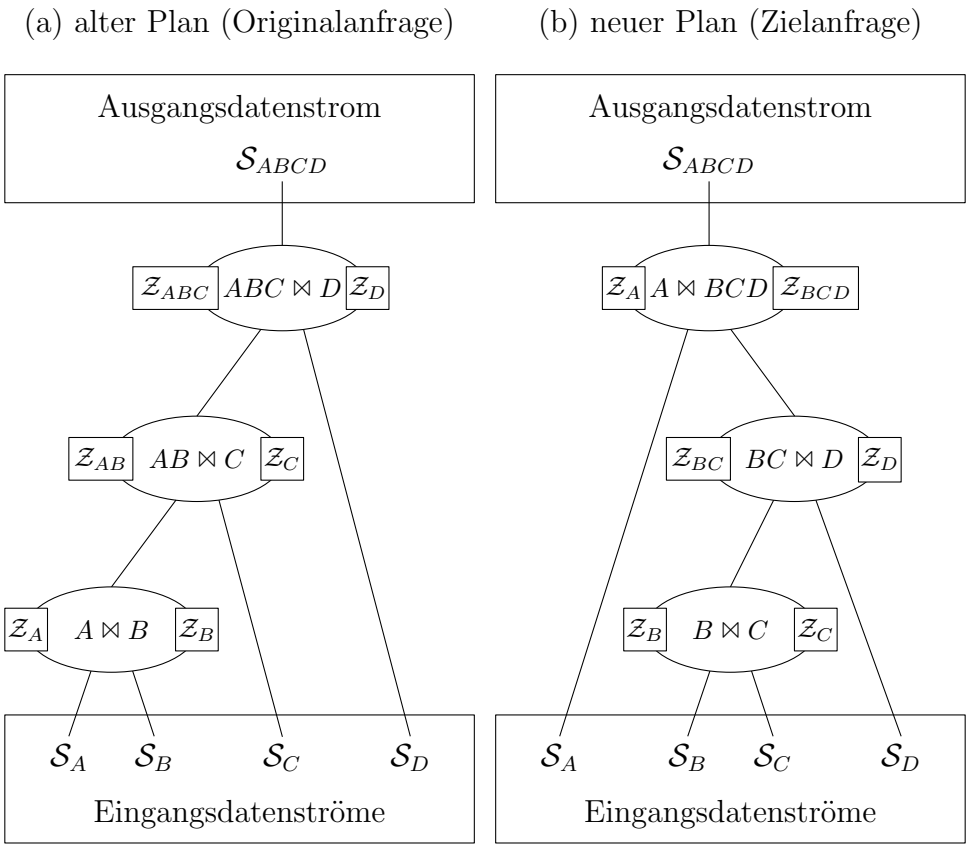


Abbildung 3.1: Beispielszenario für die Migration in CAPE (Quelle: [ZRH04])

vorhanden. Alle anderen Zustände ($\mathcal{Z}_{AB}$, $\mathcal{Z}_{ABC}$, $\mathcal{Z}_{BC}$, $\mathcal{Z}_{BCD}$) kommen nur in einem der Pläne vor. Für den Fall, dass im neuen Plan der zweite Verbundoperator $\mathcal{S}_{BC}$ mit $\mathcal{S}_A$ statt $\mathcal{S}_D$ vereint würde, wäre auch $\mathcal{Z}_{ABC}$ ein Zustand für den Zustandstransfer.

Zustandstransfer („state moving“): Nachdem die Zustände für den Zustandstransfer identifiziert wurden, werden die Tupel vom alten in den neuen Plan kopiert. Da CAPE für zentralisierte Szenarien ausgelegt ist, wird die Technik der gemeinsamen Zustände angewendet. Ein neuer Zeiger wird auf den ältesten Wert im Zustand des alten Operators gesetzt. Anschließend werden die Referenzen des alten Operators auf den Zustand gelöscht. Somit wird zeit- und speichersparend der Zustand für den neuen Plan zugänglich. Im Beispiel werden die Zustände $\mathcal{Z}_A$, $\mathcal{Z}_B$, $\mathcal{Z}_C$, $\mathcal{Z}_D$ transferiert.

Zustandsberechnung („state recomputing“): Nachdem alle übertragbaren Zustände den neuen Operatoren zugeordnet und von den alten Operatoren entkoppelt wurden, werden die verbleibenden, leeren Zwischenzustände berechnet. Dafür werden die inneren Zustände des entsprechenden Vorgängeroperators ausgelesen und der fehlende Zwischenzustand daraus generiert. Im dargestellten Beispiel sind die Zwischenzustände $\mathcal{Z}_{BC}$ und $\mathcal{Z}_{BCD}$ zu berechnen. Für den Zustand $\mathcal{Z}_{BC}$ werden also $\mathcal{Z}_B$ und $\mathcal{Z}_C$ ausgelesen. Unter Verwendung der Operatorlogik ($B \bowtie C$) wird dann $\mathcal{Z}_{BC}$ gebildet und gespeichert. Analog wird dies für $\mathcal{Z}_{BCD}$ durchgeführt.

Abschließend wird die alte Box von allen Eingangs- und Ausgangswarteschlangen entkoppelt und ihre Operatoren werden aus dem System entfernt. Die Migration ist damit beendet und die Verarbeitung wird wieder gestartet. Danach müssen zunächst alle Datentupel verarbeitet werden, die sich während der Migration in den Eingangswarteschlangen gesammelt haben, um zur aktuellen Zeit aufzuschließen. Erst wenn alle Eingangswarteschlangen das erste Mal leer sind, wurden alle angestauten Werte verarbeitet, und die Anfrage produziert ab dann aktuelle Ergebnisse.

Für MS können folgende Nachteile festgestellt werden:

(a) Bei MS wird von einer semantischen Äquivalenz zwischen altem und neuem Plan ausgegangen. Dies bedeutet, dass Änderungen von Algorithmen oder Parametern (z. B. Fenstergröße) nicht vorgesehen sind, wodurch sich das Anwendungsfeld von MS einschränkt.

(b) Mit MS werden während der Migration keine Ergebnisse produziert. In [ZRH04, Zhu06] wird dieser Effekt „Output Silence“ genannt. Diese Ergebnisse werden am Ende der Migration mit einer Verzögerung erzeugt und ausgegeben. Eine Anwendung in einer Echtzeitumgebung ist dadurch meistens nicht möglich.

(c) Die Verarbeitung der angestauten Eingangswerte am Ende der Migration führt zu einer erhöhten Systemlast. Auch dieses Verhalten kann in Echtzeitumgebungen kritisch sein.

(d) In verteilten Anwendungen ist das Prinzip der gemeinsamen Zustände nicht anwend-

bar. Zustände müssen zeitaufwendig[2] über das Netzwerk kopiert werden.

(e) Für die Migration werden alle Datenwerte der ausgewählten Zustände übertragen. Das ist für die Korrektheit von MS notwendig. Andere Verfahren können die Anzahl der Datenwerte für die Migration und somit die Transferzeit reduzieren.

(f) Weiterhin ist MS speziell auf Bäume von Verbundoperatoren zugeschnitten. Für die Anwendung auf Anfragen, die andere Operatortypen (z. B. Projektion, Selektion und Aggregation) enthalten, wurden Probleme und Lösungen in [Zhu06] angedeutet, ein allgemeines Konzept ist aber nicht erkennbar. Möglicherweise ist für jeden Operatortyp und jede Operatorkombination eine Sonderlösung notwendig.

Die Probleme von MS wurden ebenfalls in [YKPS07] und [Krä07] diskutiert. Unter der Bezeichnung GMS (Generalized Moving States) wurde in [YKPS07] eine Verallgemeinerung für MS vorgestellt. Bei GMS werden alter und neuer Plan jeweils als Black Box behandelt, d. h. die innere Struktur des Plans ist für die Migration nicht zugänglich. Nur die Eingangs- und Ausgangsströme können für die Migration benutzt werden. Das bedeutet auch, dass die Migrationsschritte in der beschriebenen Art und Weise nicht durchführbar sind. Stattdessen werden nach dem Deaktivieren der Ein- und Ausgangsaktivitäten alle Datentupel aus den inneren Zuständen des alten Plans extrahiert. Für diesen Prozess muss eine entsprechende Schnittstelle vorhanden sein. Eine Alternative ist das separate Speichern aller (vergangenen) Datenstromwerte, was in einem DSMS nicht sinnvoll erscheint. Alle extrahierten Zustandswerte werden in ihrer ursprünglichen zeitlichen Reihenfolge in die Eingangswarteschlangen eingefügt, d. h. die ältesten Werte zuerst. Um die inneren Zustände zu rekonstruieren, werden danach alle übertragenen Werte verarbeitet, wobei die entstehenden Ausgabewerte ignoriert werden. Sobald alle Werte verarbeitet wurden, ist der innere Zustand im neuen Plan vollständig und die Migration beendet. Der alte Plan wird verworfen und die Eingangswarteschlangen werden wieder mit dem entsprechenden Eingang verbunden. Auch hier müssen angestaute Datentupel verarbeitet werden, um zur aktuellen Zeit aufzuschließen. Es entsteht eine Ausgabeverzögerung. Die Migrationsdauer ist in [YKPS07] beschrieben als die Zeit, die für die Verarbeitung der alten Werte im neuen Plan (zur Erstellung der inneren Zustände) benötigt wird. Eine Zustandstransferdauer wird aufgrund der lokalen Ausführung nicht berücksichtigt. Außerdem wird die Zeit zur Verarbeitung der angestauten neuen Werten nicht in die Migrationsdauer eingerechnet.

Obwohl diese Variante mit beliebigen Operatoren verwendbar ist, bleiben die größten Nachteile von MS bestehen. Während der Migration werden keine Ergebnisse produziert und nach der Migration werden angestaute Eingangswerte mit maximalem Durchsatz berechnet und ausgegeben. Das Prinzip der gemeinsamen Zustände lässt sich bei GMS nicht anwenden, die Daten müssen extrahiert und wieder in die Warteschlangen eingefügt werden. Dieser Prozess kostet mehr Zeit als Referenzänderungen im Arbeitsspeicher.

[2] im Vergleich zum Ändern der Referenzen im Arbeitsspeicher

Sollte die Strategie in einer verteilten Umgebung angewendet werden, muss zusätzlich die Transferzeit berücksichtigt werden. Der Vorteil gegenüber MS ist jedoch, dass GMS allgemein anwendbar ist, das Problem (f) entfällt somit. Die Probleme (a) bis (e) bleiben auch mit GMS bestehen.

3.1.2 Die Migrationsstrategie „Parallel Track“

Um den Nachteil, keine Ergebniswerte während der Migration auszugeben, zu kompensieren, wurde in [ZRH04, Zhu06] die Migrationsstrategie „Parallel Track“ (PT) beschrieben, eine Umsetzung der Parallelmethode. Dabei wird der neue Plan parallel zur Verarbeitung des originalen Plans mit Datenstromwerten versorgt und ausgeführt. Eine Zustandsübertragung aus dem alten in den neuen Plan wird nicht durchgeführt.

Mit dem Migrationsbeginn ($t_{M,start}$) wird die Ausführung der alten Box kurz pausiert, um die Ein- und Ausgangswarteschlangen beider Boxen miteinander zu verbinden, d. h. mittels gemeinsamer Eingangswarteschlangen werden beide Pläne mit Werten versorgt und die Ergebnisse werden an die gleiche Ausgangswarteschlange[3] gesendet.

Um das Ende der Migration zu bestimmen, findet während der Migration für alle Datenstromelemente eine Fallunterscheidung anhand ihrer Zeitstempel[4] statt. Alle Datenstromtupel, die vor $t_{M,start}$ erzeugt wurden oder zusammengesetzte Werte, die wenigstens ein Sub-Tupel enthalten, dessen Zeitstempel vor dem Migrationsbeginn liegt ($ts_{min} < t_{M,start}$), werden als „alt“ bezeichnet. Alle anderen Werte werden als „neu“ bezeichnet, d. h. alle Eingangswerte, die nach $t_{M,start}$ erzeugt wurden oder zusammengesetzte Werte, die ausschließlich neue Sub-Tupel enthalten. Für zusammengesetzte Datenstromelemente ist die Charakterisierung, ob alt oder neu, unabhängig von dessen Zeitstempel.

$\mathcal{D}_A$	$\mathcal{D}_B$	$\mathcal{D}_{AB}$	$\mathcal{D}_{AB}.ts$
alt	alt	alt	$max(\mathcal{D}_A.ts, \mathcal{D}_B.ts) < t_{M,start}$
alt	neu	alt	$\mathcal{D}_B.ts > t_{M,start}$
neu	alt	alt	$\mathcal{D}_A.ts > t_{M,start}$
neu	neu	neu	$max(\mathcal{D}_A.ts, \mathcal{D}_B.ts) > t_{M,start}$

Tabelle 3.1: Alt-Neu-Markierung und Zeitstempelzuweisung bei Verbundoperationen von Datenstromwerten bei der Parallel Track Strategie am Beispiel $\mathcal{S}_A \bowtie \mathcal{S}_B$

Die möglichen Kombinationen sind in Tabelle 3.1 aufgeführt. Ausgangspunkt sind jeweils zwei Tupel $\mathcal{D}_A$ und $\mathcal{D}_B$ aus den Datenströmen $\mathcal{S}_A$ bzw. $\mathcal{S}_B$. Diese werden durch einen Verbundoperator ($\mathcal{S}_A \bowtie \mathcal{S}_B$) zu einem Tupel $\mathcal{D}_{AB}$ verbunden. Die ersten drei Spalten beinhalten die Markierungen der Eingangstupel und des Ergebnistupels. Die ersten

[3] Eingangswarteschlange des nachfolgenden Operators

[4] Die Zeitstempelvergabe wurde in Abschnitt 2.2.1 beschrieben.

drei Fälle resultieren in einem als alt markierten Ergebnistupel, da jeweils mindestens ein altes Sub-Tupel enthalten ist. In der vierten Spalte ist der Zeitstempel des Ergebnistupels im Vergleich zu $t_{M,start}$ angegeben. Er wird immer aus $\mathcal{D}_{AB}.ts = max(\mathcal{D}_A.ts, \mathcal{D}_B.ts)$ gebildet. Für das erste Beispiel ist der Zeitstempel kleiner als $t_{M,start}$, da beide Eingangstupel davor erzeugt wurden. Bei allen anderen Fällen liegt er über $t_{M,start}$, weil wenigstens ein Sub-Tupel danach erzeugt wurde.

Während der Migration werden alte Datenstromwerte durch neue Werte aus den Zuständen verdrängt und verschwinden somit allmählich aus dem alten Plan. Der neue Plan kann keine alten Datentupel enthalten, da er mit leeren Zuständen zu arbeiten beginnt und ausschließlich Werte erzeugt, die nur neue Sub-Tupel enthalten. Die Migration endet, wenn alle alten Datenstromelemente entfernt wurden und nur noch neue in der alten Box vorhanden sind. Da somit die Zustände beider Pläne identisch sind, kann die alte Box verworfen werden, und die Migration ist damit abgeschlossen.

Die Anforderung, alte von neuen Tupeln zu unterscheiden, erfordert zusätzliche Mechanismen während der Migration, um zunächst zu entscheiden, welche Tupel alt und welche neu sind. Basierend darauf ist fortlaufend festzustellen, wann Zustände des alten Plans frei von alten Tupeln sind. Mit Hilfe einer Überwachungsfunktion für den alten Plan wird dies überprüft. Deshalb ist ein zusätzlicher Aufwand im Vergleich zum normalen Betrieb der Operatoren einzukalkulieren.

Mit PT erhalten beide Boxen Eingangsdaten und produzieren Ergebnisse. Um der Anforderung zu genügen, keine Ergebniswerte doppelt an die Ausgangswarteschlange auszugeben, müssen die Ausgänge beider Boxen gefiltert werden. Die alte Box gibt alle alten Werte aus, also diejenigen, die wenigstens ein altes Sub-Tupel enthalten. Die neue Box gibt entsprechend nur neue Werte aus. Um das zu realisieren, werden während der Migration zwei spezielle Implementierungen von Verbundmethoden verwendet – eine für Operatoren der alten Box und eine für Operatoren der neuen Box. Zudem wird in der Verbundmethode im letzten Operator der alten Box eine Filterung durchgeführt. Sie erzeugt keine Ergebnisse, die ausschließlich neue Sub-Tupel enthalten würden, da diese durch die neue Box generiert werden. Die neue Box speichert alle produzierten Ergebnisse in einem temporären Puffer und gibt die Werte nach dem Verwerfen der alten Box an die Ausgangswarteschlange aus.

Die Verwendung des temporären Puffers soll der Erhaltung der zeitlichen Reihenfolge von Ergebniswerten dienen, basiert jedoch auf einer falschen Schlussfolgerung, dass Zeitstempel von Ergebnissen der neuen Box immer größer sind als die der alten Box[5]. Somit erzeugt das Puffern der Ergebnisse der neuen Box den umgekehrten Effekt und zerstört die zeitliche Reihenfolge des Datenstroms.

Die Migrationsdauer für PT beträgt für Verbundoperatorbäume mit zeitlich geordne-

[5] Aufgrund der Unabhängigkeit zwischen Zeitstempelvergabe im Verbundoperator und Alt-Neu-Markierung der Ergebnisse haben alle ausgegebenen Datenstromwerte der alten Box einen größeren Zeitstempel als *zuvor* produzierte Ergebnisse der neuen Box. Zum Beispiel wenn $\{\mathcal{D}_a, \mathcal{D}_b\} \in \mathcal{S}_X$ und $\{\mathcal{D}_c, \mathcal{D}_d\} \in \mathcal{S}_Y$ und $\mathcal{D}_a.ts < t_{M,start} < \mathcal{D}_b.ts < \mathcal{D}_c.ts < \mathcal{D}_d.ts$, dann $\mathcal{D}_{bc}.ts < \mathcal{D}_{ad}.ts$, da $\mathcal{D}_c.ts < \mathcal{D}_d.ts$, wobei $\mathcal{D}_{bc}$ von der neuen Box generiert wird und $\mathcal{D}_{ad}$ von der alten.

ten Eingangsströmen, zeitbasierten Fenstern und bei Verwendung einer globalen zeitbasierten Fenstergröße w laut [ZRH04] zwischen w und $2w$. Abhängig sind diese Werte, von der Anzahl der Ebenen h des Operatorbaums in der alten Box, da die Migration bei PT endet, wenn alle alten Werte aus der alten Box entfernt sind.

Bei einer einstufigen Anfrage ($h = 1$) entspricht die Migrationsdauer w. Dies ist plausibel, stellt man sich einen einzelnen Verbundoperator vor, so benötigt man eine Zeit von w bis alle alten Werte seiner inneren Zustände durch neue Werte ersetzt sind. Die Migration entspricht seinem Anlaufverhalten und entsprechend die Migrationsdauer seiner Anlaufzeit.

Für $h > 1$ ermittelt [ZRH04] eine Migrationsdauer von $T_M = 2 * w$, diese Zeit ist jedoch nicht nachvollziehbar. Der Inhalt der Zustände hängt im Wesentlichen von den Mechanismen der Verbundoperatoren ab. Werte werden basierend auf ihren Zeitstempeln und der eingestellten Fenstergröße verworfen. Weitere beeinflussende Eigenschaften sind die Frequenz der eingehenden Ströme und der Zeitpunkt, wann die Operatorzustände aktualisiert werden. Nimmt man eine realistische Konfiguration der Verbundoperatoren an, bei der die Periodendauern der eingehenden Datenströme kleiner sind als die Fenstergrößen sowie eine Aktualisierung der Zustände mit jedem neuen Eingangswert, dann ist die Methode zum Verwerfen von Werten aus dem Fenster zu betrachten, um die Migrationsdauer für PT zu bestimmen.

In CAPE wird in Verbundoperatoren dazu die Differenz aus dem Zeitstempel des eingehenden Tupels[6] und dem minimalen Zeitstempel des Tupels im Zustand des Operators gebildet und mit der Fenstergröße verglichen. Ist die Differenz größer als die Fenstergröße ($max(ts_{input}) - min(ts_{state}) > w$), wird das Tupel aus dem Zustand verworfen. Das bedeutet bei der Verwendung einer globalen Fenstergröße, dass alle Sub-Tupel aller Werte in allen Zuständen immer innerhalb der globalen Fenstergröße liegen, egal wie viele Ebenen der Operatorbaum enthält. Somit dauert die Migration auch für $h > 1$ genauso lange wie w.

Würde das Verwerfen nur mit den Zeitstempeln der Tupel und ohne Berücksichtigung der Zeitstempel der Sub-Tupel stattfinden ($max(ts_{input}) - max(ts_{state}) = ts_{input} - ts_{state} > w$), so werden auf jeder Ebene Sub-Tupel innerhalb einer Spanne von w erhalten. Die Migrationsdauer ist dann von der Anzahl der Ebenen abhängig, d. h. $T_M = h * w$. Für das CAPE-System ist dies jedoch nicht der Fall.

Unter den angegebenen Randbedingungen, d. h. zeitlich geordnete Eingangsströme, zeitbasierte gleitende Fenster, eine globale Fenstergröße und ausschließlich Verbundoperatoren, dauert die Migration mit PT also w. Sobald sich eine dieser Bedingungen ändert, ist die Migrationsdauer neu zu bewerten und gegebenenfalls nicht bestimmbar. Als Beispiel kann man sich einen Aggregationsoperator mit anzahlbasiertem Fenster hinter einem Filteroperator vorstellen. Da die Ausgabe des Filters von den Attributwerten seines eingehenden Datenstroms abhängt, ist während der Migration nicht vorauszusagen, wann alle alten Datenwerte aus dem Fenster des Aggregationsoperators im alten Plan

[6] entspricht dem maximalen Zeitstempel aller enthaltenen Sub-Tupel

verdrängt wurden. Dadurch ist auch das Migrationsende nicht vorhersagbar.

Nachfolgende Probleme lassen sich für PT identifizieren:

(a) Bei PT wird von einer semantischen Äquivalenz zwischen altem und neuem Plan ausgegangen. Dies bedeutet, dass Änderungen von Algorithmen oder Parametern (z. B. Fenstergröße) nicht vorgesehen sind, wodurch sich das Anwendungsfeld von PT einschränkt.

(b) Die Migrationsdauer von PT entspricht mindestens der größten Anlaufzeit der einzelnen Operatoren in der neuen Box, d. h. wenn beispielsweise die größte Fenstergröße 30 Stunden beträgt, dann wird die Migration mindestens 30 Stunden benötigen. Für komplexe Anfragen kann die Migrationsdauer deutlich höher sein. Mögliche Einflussfaktoren sind die Struktur der Anfrage, die Charakteristik der Eingangsdatenströme, die Eigenschaften der verwendeten Operatoren und die Fenstertypen.

(c) Bei der Verwendung anzahlbasierter Fenster in aperiodischen Datenströmen ist die Anlaufzeit der Operatoren und somit die Migrationsdauer nicht bestimmbar.

(d) PT ist nicht für Anfragen anwendbar, die wachsende Fenster enthalten, deren festes Ende vor dem Migrationsbeginn liegt, da vergangene Datenstromwerte nicht in die Zustände der neuen Anfrage aufgenommen werden können.

(e) Wie bei MS werden für PT gemeinsame Warteschlangen benutzt. Dieses Konzept ist nur in zentralisierten Lösungen anwendbar, jedoch nicht in verteilten Umgebungen.

(f) Während der Migration ist eine Überwachungsmethode für die alte Box notwendig, um festzustellen, wann die einzelnen Operatoren frei von alten Tupeln sind. Dies erfordert eine direkte Zugriffsmöglichkeit auf jeden Operator der alten Box und einen zusätzlichen Verarbeitungsaufwand für die Überwachungsmethode.

(g) Durch das temporäre Speichern der Ergebnisse der neuen Box werden Ergebnisse verzögert und die zeitliche Reihenfolge der Ergebniswerte wird vertauscht. Dies widerspricht den definierten Anforderungen und ist besonders in Echtzeitanwendungen nicht akzeptabel.

(h) Das temporäre Speichern erfordert zusätzlichen Speicher, der ansonsten nicht benötigt wird und unter Umständen nicht verfügbar ist.

(i) Da während der Migration ausschließlich Ergebnisse mit alten Sub-Tupeln ausgegeben werden, alte Werte aber allmählich aus den Zuständen verworfen werden, ist an der Ausgangswarteschlange ein abnehmender Durchsatz zu beobachten.

(j) Die plötzliche Freigabe der Ergebnisse der neuen Box am Ende der Migration erzeugt eine hohe Last für nachfolgende Operatoren. Unter Umständen wird dadurch die Bearbeitung neuer Datenstromwerte verzögert.

(k) Für PT werden während der Migration spezielle Verbundmethoden in der alten und neuen Box verwendet, um die zusätzlichen Funktionen, z. B. Filtern, zu realisieren. Einerseits ist dies keine generische Lösung, sondern nur bei Verbundoperatoren anwendbar. Zudem ist ein sicherer Wechsel zum normalen Algorithmus am Ende der Migration zu gewährleisten. Wie der Algorithmenaustausch am Ende der Migration funktioniert, ist jedoch nicht beschrieben.

(l) Wie MS ist PT speziell auf Bäume von Verbundoperatoren zugeschnitten. Für Anfragen, die zusätzlich zu den Verbundoperatoren zustandsfreie Operatoren wie Projektion und Selektion enthalten, wurde in [Zhu06] die Anwendbarkeit gezeigt. Deutlich schwieriger ist laut [Zhu06] die Verwendung von PT bei Kombinationen von Verbundoperatoren mit anderen zustandsbehafteten Operatoren, z. B. Gruppierungs- und Aggregationsoperatoren. Lösungen für einfache Beispiele wurden in [Zhu06] gezeigt, wobei zum Teil zusätzliche Operatoren während der Migration benötigt werden. Wie praktikabel diese Konzepte bei der Migration von komplexen Anfragen sind, wurde nicht diskutiert.

Auch für PT wurden die Probleme in [YKPS07] und [Krä07] diskutiert und eine Verallgemeinerung in [YKPS07] vorgeschlagen. Die Generalized Parallel Track (GPT) Strategie behandelt alten und neuen Plan ebenso jeweils als Black Box. Bei Migrationsbeginn werden wie bei PT die Eingangsdatenströme mit beiden Boxen verbunden. Die Ergebnisse während der Migration werden aber nur von der alten Box an die Ausgangswarteschlange gesendet. Die Ergebnisse der neuen Box werden an eine temporäre Datensenke übertragen, wo sie direkt verworfen werden. Am Ende der Migration, werden die alte Box und die temporäre Datensenke entkoppelt und verworfen. Die Ausgangswarteschlange erhält fortan die Datenstromwerte von der neuen Box.

Da während der Migration ausschließlich der alte Plan Ergebnisse an nachfolgende Systemelemente weiterleitet, sind keine Methoden zum Filtern oder Vereinen der Ergebnisse nötig. Dies sorgt dafür, dass alle Ergebniswerte in der richtigen temporalen Reihenfolge ausgegeben werden. Ebenso kann auf eine Alt-Neu-Markierung und eine entsprechende Auswertung der Datenstromwerte verzichtet werden.

Für einen Baum von Verbundoperatoren wie im Migrationsbeispiel (Abbildung 3.1) mit zeitlich geordneten Eingangsdatenströmen, zeitbasierten gleitenden Fenstern und einer globalen Fenstergröße w entspricht die Migrationsdauer mit GPT w. Sind andere zustandsbehaftete Operatoren in der Anfrage enthalten, bleiben die Probleme von PT bezüglich der Migrationsdauer bestehen. Wachsende Fenster, deren festes Ende vor dem Migrationsbeginn liegt, lassen sich generell nicht mit PT oder GPT migrieren.

Insgesamt können durch GPT die Nachteile (g) bis (l) behoben werden. Die Probleme (a) bis (e) sind weiterhin vorhanden. Besonders nachteilig ist die lange Migrationsdauer bei großen Fenstergrößen oder bei ungünstigen Operatorkombinationen. Das Problem (f) tritt zumindest bei Bäumen von Verbundoperatoren mit zeitlich geordneten Eingangsdatenströmen, zeitbasierten Fenstern und einer globalen Fenstergröße nicht auf.

Bei Konfigurationen mit beliebigen Operatoren ist jedoch zu erwarten, dass Problem (f) bestehen bleibt.

3.1.3 D-CAPE

D-CAPE (Distributed Continuous Adaptive Processing System [CAP09, SR04]) ist eine verteilte Architektur auf Basis des CAPE-Systems. Laut [ZRH04] wurden MS und PT für die Verwendung in D-CAPE implementiert. Es ist jedoch nicht ersichtlich, wie einige Funktionen, z. B. geteilte Zustände oder geteilte Warteschlangen, in der verteilten Umgebung von D-CAPE funktionieren oder realisiert wurden. Es ist anzunehmen, dass Zustände zwischen Knoten kopiert werden, was die Vorteile der genannten Mechanismen neutralisiert. Besonders für MS ist durch die Laufzeit im Netzwerk zwischen den Knoten mit einer Verlängerung der Migration zu rechnen.

In [SR04] werden für die Optimierung von Anfragen unter anderem Methoden zum Verschieben von Operatoren zwischen Verarbeitungsknoten vorgestellt. Für die Migration von Operatoren wird ein Mehrschrittverfahren gezeigt, das MS ähnelt und wie folgt ausgeführt wird. Nach der Auswahl des zu verschiebenden Operators und des Zielknotens werden am Zielknoten zunächst die ausgehende Verbindung und danach die eingehenden Verbindungen erstellt und am aktuellen Knoten getrennt. Der Originaloperator wird angehalten, wenn alle verbundenen Eingangswarteschlangen leer sind. Als nächstes wird bei zustandsbehafteten Operatoren der Zustand extrahiert und an den Knoten des neuen Operators transferiert. Nachdem der neue Operator mit dem Zustand initialisiert wurde, wird die Verarbeitung fortgesetzt. Beim Umschalten vom Ausgang des alten Operators auf den des neuen Operators wird darauf geachtet, dass zunächst alle Ergebnisse des alten Operators ausgegeben werden und danach die des neuen, um zeitliche Vertauschungen zu vermeiden.

Da das Verfahren der MS-Strategie ähnelt, sind auch die gleichen Nachteile zu erwarten. Obwohl der Output-Silence-Effekt nicht explizit in [SR04] diskutiert wird, ist davon auszugehen, dass er auftritt. Zudem sind die Synchronisation beim Verbinden der Eingänge und beim Umschalten des Ausgangs die kritischsten Schritte im gesamten Ablauf, da Datenverlust oder Beeinflussung der Anfragenbearbeitung zu vermeiden sind. Diese Schritte wurden jedoch nicht im Detail diskutiert, da der Schwerpunkt von [SR04] auf der Verteilungsplanung und -optimierung liegt.

3.2 PIPES

Basierend auf den Migrationsstrategien MS und PT wurden zwei Lösungen für das Datenstromsystem PIPES vorgeschlagen – GenMig und HybMig. Beide Strategien werden nachfolgend im Einzelnen erläutert.

3.2.1 Die Migrationsstrategie „GenMig“

Die Migrationsstrategie „GenMig“ (General Migration) [Krä07] ist eine Realisierung des Parallelprinzips und basiert auf PT. GenMig wurde beschrieben für zwei in [Krä07] vorgestellte Ansätze zur Zeitstempelbehandlung in Datenstromsystemen – einerseits für den in PIPES bevorzugten Zeitintervallansatz (time-interval approach, TIA) und des Weiteren für den Positiv-Negativ-Ansatz (positive-negative approach, PNA), der beispielsweise auch in STREAM [ABB+04] oder Nile [HGA+07] verwendet wird. Nachfolgend wird der allgemeine Ansatz von GenMig für den PNA beschrieben, da der PNA dem in dieser Arbeit genutzten Ansatz[7] am meisten ähnelt. Der Vollständigkeit wegen werden auch die Lösungen für die Verwendung des TIA angegeben.

Ein Zugriff auf einzelne Operatoren im alten oder neuen Plan ist für GenMig nicht notwendig, sodass beide Pläne jeweils als Black Box betrachtet werden. Die Migration beginnt mit der Beobachtung und Analyse aller eingehenden Datenströme. Beim PNA wird für jeden Eingangsdatenstrom der Zeitstempel des aktuellen Datentupels ermittelt. Beim TIA werden die entsprechenden Startzeitstempel der Tupel erfasst. Sobald für jeden Eingangsstrom dieser Wert bekannt ist, wird die Ausführung des alten Plans pausiert.

Danach wird basierend auf den erfassten Zeitstempeln der Zeitpunkt T_{split} berechnet. Dieser Zeitpunkt markiert das Ende der Migration. Mit Hilfe von T_{split} wird entschieden, welche Eingangswerte an die alte oder neue Box gesendet werden und ob Ergebniswerte von der alten Box oder von der neuen Box an die Ausgangswarteschlange übermittelt werden. Ein Ergebnis der alten Box wird weitergeleitet, falls dessen Zeitstempel vor T_{split} liegt. Mit Überschreiten von T_{split} werden die Ergebnisse der neuen Box übertragen. Sind alle Zeitstempel aller Eingänge größer als T_{split}, ist die Migration beendet.

Für das in Abbildung 3.1 dargestellte Beispiel mit der globalen Fenstergröße w ist $T_{split} = max(\{ts_{\mathcal{S}_i}\}) + w + \varepsilon$, wobei $\{ts_{\mathcal{S}_i}\}$ die Menge der aktuellen Zeitstempel aller n Eingangsströme ist $(\forall i \in \{1, \ldots, n\})$ und ε ein Wert, der kleiner ist, als die Zeitgranularität der Zeitstempel der Datenstromwerte im System. Die Verwendung von ε dient zur Vermeidung der unerwünschten Situation, dass der Zeitstempel eines Datenstromwertes genau T_{split} entspricht.

Nach der Berechnung von T_{split} werden zusätzliche Hilfsoperatoren vor und hinter die zu migrierenden Pläne eingefügt. Zwischen jeden Eingangsstrom und die alte und neue Box kommt ein Fork-Operator, der den jeweiligen Eingangsstrom entsprechend des Zeitstempels des ankommenden Datenwertes behandelt. Beim PNA wird jeder Eingangswert an die neue Box weitergeleitet und falls $ts < T_{split}$ ebenfalls an die alte Box. Bei der Verwendung des TIA werden Datenstromwerte zerlegt und mit modifizierten Zeitintervallen an die alte und neue Box gesendet.

Weiterhin werden zwischen die Boxen und jede Ausgangswarteschlange beim PNA zwei Hilfsoperatoren eingefügt – ein Filteroperator am Ausgang der neuen Box und einen Vereinigungsoperator, um die gefilterten Werte und die Ausgabe der alten Box zu

[7] entspricht „raw stream“ in [Krä07]

einem Ausgangsstrom zusammenzufassen. Für den Filter gilt dabei, dass alle Werte mit einem Zeitstempel $ts < T_{split}$ verworfen werden, da diese von der alten Box generiert werden. Alle Werte mit $ts > T_{split}$ bleiben erhalten und werden weitergeleitet. Beim TIA wird anstatt der Filter- und Vereinigungsoperatoren ein Fuse-Operator verwendet. Dieser setzt die zuvor zerlegten Datenstromwerte wieder zusammen und kehrt somit die Funktion des Fork-Operators um. Der Fuse-Operator ist zustandsbehaftet.

Nachdem alle Hilfsoperatoren eingefügt wurden, wird die Ausführung des alten Plans fortgesetzt und die des neuen Plans begonnen. Während der Migration werden weiterhin die Zeitstempel der eingehenden Werte beobachtet. Die Migration endet, wenn $min(\{ts_{\mathcal{S}_i}\}) > T_{split}$, d. h. wenn alle aktuellen Datenstromwerte der Eingangsströme nach T_{split} erzeugt wurden. Da mit diesem Zeitpunkt die alte Box nicht mehr notwendig ist, wird ihre Ausführung gestoppt und ihr Anfrageplan wird aus dem System entfernt. Die neue Box wird pausiert, um die Hilfsoperatoren zu entfernen, da sie nach der Migration nicht mehr benötigt werden. Alle Ein- und Ausgangsströme sind dann direkt mit dem neuen Plan verbunden und die Ausführung kann fortgesetzt werden.

Das bis hierhin beschriebene Modell ist ausschließlich auf Anfragen mit zeitbasierten Fenstern mit einer globalen Fenstergröße anwendbar. Ursache ist die Zeit T_{split}, die von der globalen Fenstergröße abhängt. Sobald andere zustandsbehaftete Operatoren oder andere Fenstertypen in einer Anfrage verwendet werden, ist die Vorhersage der Migrationsdauer und somit auch die Vorhersage von T_{split} schwierig oder nicht möglich.

Zur Anwendung von GenMig mit Anfragen, die beliebige Operatoren und Fenstertypen enthalten, wurde in [Krä07] eine Erweiterung vorgeschlagen. Zusätzlich zur Erfassung der Menge der aktuellen Zeitstempel aller Eingangsdatenströme werden vor Migrationsbeginn die Endzeitstempel von Elementen, die in die alte Box gesendet werden, überprüft. Die Menge $\{ts_{E_i}\}$ $\forall i, n \in \mathbb{N}^+$ und $i \leq n$ umfasst den jeweils maximalen Endzeitstempel aller n Eingangsdatenströme. Das Migrationsende ist dann $T_{split} = max(\{ts_{E_i}\}) + \varepsilon$.

Mit GenMig werden viele Probleme von PT vermieden, z. B. der kontinuierlich sinkende Durchsatz oder das Vertauschen der zeitlichen Reihenfolge von Ergebnistupeln. Die wesentlichen Probleme von PT und GPT ((a) bis (d)) bleiben jedoch erhalten. Zudem treten durch GenMig neue Nachteile auf. Alle Probleme sind nachfolgend zusammengefasst.

(a) Wie bei PT wird bei GenMig von einer semantischen Äquivalenz zwischen altem und neuem Plan ausgegangen. Dies bedeutet, dass Änderungen von Algorithmen oder Parametern (z. B. Fenstergröße) nicht vorgesehen sind, wodurch sich das Anwendungsfeld von GenMig einschränkt.

(b) Die Migrationsdauer von GenMig entspricht mindestens der größten Anlaufzeit der einzelnen Operatoren in der neuen Box. Für komplexe Anfragen kann die Migrationsdauer deutlich höher sein. Mögliche Einflussfaktoren sind die Struktur der Anfrage, die Charakteristik der Eingangsdatenströme, die Eigenschaften der verwendeten Operatoren und die Fenstertypen.

(c) Bei der Verwendung anzahlbasierter Fenster in aperiodischen Datenströmen ist die Anlaufzeit der Operatoren und somit die Migrationsdauer nicht bestimmbar. Dies bedeutet, dass sich T_{split} nicht ermitteln lässt und somit GenMig nicht angewendet werden kann.

(d) GenMig ist nicht für Anfragen anwendbar, die wachsende Fenster enthalten, deren festes Ende vor dem Migrationsbeginn liegt, da vergangene Datenstromwerte nicht in die Zustände der neuen Anfrage aufgenommen werden können.

(e) Während der Migration ist die Verwendung von zusätzlichen Operatoren (Fork, Fuse oder Filter und Vereinigung) notwendig. Dies stellt einen zusätzlichen Aufwand dar, da Datenwerte von diesen Operatoren zusätzlich verarbeitet werden müssen. Für den Fuse-Operator wird zusätzlicher Speicher zum Puffern der Ausgangswerte benötigt, welcher unter Umständen nicht verfügbar ist.

(f) Das Einfügen und Entfernen von zusätzlichen Operatoren beeinflusst die Ausführung der laufenden Anfrage, da diese jeweils kurz angehalten wird. Während der Unterbrechung werden Eingangswerte gepuffert und Ergebnisse verzögert.

(g) Die Realisierung von ε erfordert die Existenz einer minimalen Zeitgranularität für Datenwerte im System. Es ist unklar, wie in Systemen ohne eine solche minimale Zeitgranularität vorzugehen ist besonders im Zusammenhang mit aperiodischen Datenströmen.

(h) Die vorgeschlagene Verallgemeinerung lässt sich nur beim TIA anwenden, da beim PNA keine Endzeitstempel existieren.

Zusammenfassend bleibt festzuhalten, dass bei GenMig, wie auch bei PT und GPT, mit einer langen Migrationsdauer zu rechnen ist. Der Hauptvorteil im Vergleich zu PT ist die gleichbleibende Ausgabe von Ergebnissen während der gesamten Migration sowie die Erhaltung der zeitlichen Reihenfolge des Ergebnisdatenstroms. Außerdem wird dadurch die plötzliche Freigabe von gepufferten Ergebnissen des neuen Anfrageplans vermieden. Aus diesem Grund kommt GenMig mit weniger Speicher aus, da entweder wenige (TIA) oder keine Ausgangswerte (PNA) gepuffert werden müssen. Eine wesentliche Einschränkung von GenMig ist, dass es sich in der allgemeinen Form nur in Systemen mit TIA anwenden lässt. In allen anderen Systemen bleibt der Einsatz auf Anfragen mit zeitbasierten Fenstern beschränkt.

3.2.2 Die Migrationsstrategie „HybMig“

Die Migrationsstrategie „HybMig“ (Hybrid Migration) [YKPS07] kombiniert Eigenschaften von MS und PT und realisiert demzufolge eine Mischung aus Zustandstransfer und Parallelmethode. Während der Migration werden Ergebnisse kontinuierlich ausgegeben und innere Operatorzustände werden erhalten. HybMig nutzt die gleiche Definition für

den Alt-Neu-Status von Datentupeln wie die Migrationsstrategien in CAPE. Implementiert wurde HybMig im PIPES-System.

In der einfachsten Version werden alte und neue Box parallel mit Eingangswerten versorgt. Zu Beginn der Migration wird wie bei MS ein Zustandsvergleich durchgeführt. Anschließend werden alle Zustände, die beide Boxen gemeinsam haben, der neuen Box zur Verfügung gestellt. HybMig ist ausschließlich für eine zentralisierte Anwendung beschrieben, sodass aus Effizienzgründen *gemeinsame Zustände* benutzt werden. Während einer kurzen Unterbrechung der Verarbeitung werden Ein- und Ausgangsströme mit beiden Boxen verbunden und der Zugriff auf die gemeinsamen Zustände initialisiert. Im Beispiel in Abbildung 3.1 sind dies alle Zustände der Eingangsströme ($\mathcal{Z}_A$, $\mathcal{Z}_B$, $\mathcal{Z}_C$, $\mathcal{Z}_D$). Die übrigen Zustände im neuen Plan ($\mathcal{Z}_{BC}$, $\mathcal{Z}_{BCD}$) bleiben zunächst leer und werden durch neu ankommende Werte kontinuierlich gefüllt.

Im weiteren Verlauf produzieren beide Pläne Ergebnisse. Die neue Box ist in der Lage, die meisten Alt-Neu-Kombinationen zu erzeugen. Alle Kombinationen, die nicht durch die neue Box erzeugt werden können, werden von der alten Box generiert. Im Migrationsbeispiel gilt dies für alle Ergebniswerte, bei denen die Sub-Tupel $\mathcal{D}_B$ und $\mathcal{D}_C$ *gleichzeitig* alt sind. Dies bedeutet auch, dass vom alten Plan keine Ergebnisse mit neuen Sub-Tupeln von $\mathcal{D}_B$ und $\mathcal{D}_C$ erzeugt werden dürfen.

Um eine korrekte Ausgabe zu erhalten, ist eine Filterung innerhalb der alten Box notwendig. Im Allgemeinen wird dafür eine Merkmalstabelle von Tupelsignaturen erzeugt. Eine Signatur gibt an, welche Sub-Tupel alt sein müssen und welche Sub-Tupel einen beliebigen Status haben können. Die Signaturen werden genutzt, um unerwünschte Sub-Tupel-Kombinationen in der alten Box zu vermeiden. Dies erfordert Modifikationen an den Verbundalgorithmen in der alten Box, um Tupel auf ihren Alt-Neu-Status zu überprüfen und eine geeignete Filterung durchzuführen. In Sonderfällen kann auf eine Merkmalstabelle verzichtet werden und Eingangsdatenströme werden von der alten Box entkoppelt. Ein Mechanismus zur Identifikation dieser Sonderfälle ist jedoch nicht beschrieben.

Die Realisierung für das Migrationsbeispiel ist wie folgt. Auf Grund des neuen Operators $B \bowtie C$ im neuen Plan existieren im nachfolgenden Operator keine alten Datenstromwerte im Zustand $\mathcal{Z}_{BC}$. Datenstromwerte, die alte Sub-Tupel $\mathcal{D}_B$ und $\mathcal{D}_C$ besitzen, müssen vom alten Plan generiert werden. Deshalb enthält die Merkmalstabelle einen Eintrag $[\mathcal{D}_A : \mathit{beliebig}; \mathcal{D}_B : \mathit{alt}; \mathcal{D}_C : \mathit{alt}; \mathcal{D}_D : \mathit{beliebig}]$. Die Verbundalgorithmen in den Operatoren $AB \bowtie C$ und $ABC \bowtie D$ werden daraufhin so modifiziert, dass genau diese Kombinationen erzeugt werden können, alle anderen aber nicht. Ebenso kann bei diesem Migrationsbeispiel die zweite Lösung genutzt werden, indem man die Eingangsströme $\mathcal{S}_B$ und $\mathcal{S}_C$ von der alten Box entkoppelt. Eine Modifikation der Verbundalgorithmen ist dann nicht notwendig.

Für beide Realisierungen gilt, dass die neuen Zustände im neuen Plan ausschließlich mit neuen Werten gefüllt werden. Das bedeutet, dass die Migrationsdauer der globalen Fenstergröße w entspricht, da genau nach dieser Zeit die beiden neuen Zustände ($\mathcal{Z}_{BC}$ und $\mathcal{Z}_{BCD}$) das erste Mal vollständig gefüllt sind.

Eine weiterentwickelte Version von HybMig verwendet die ungenutzte Rechenzeit des Systems, um die leeren Zwischenzustände aus den alten Werten der bekannten Zustände zu berechnen. Dieser Prozess wird „Background State Computation“ (BSC) genannt und mit einer geringeren Priorität als die normale Verarbeitung durchgeführt, um diese nicht negativ zu beeinflussen. Wie der BSC-Prozess gehandhabt wird, wenn ein neuer Datenwert eines Eingangsstroms den Hauptprozess startet, ist nicht beschrieben. Sinnvollerweise sollte BSC in einem definierten Zustand unterbrochen werden, damit es in der nächsten Pause nahtlos fortgesetzt werden kann. Details sind in [YKPS07] jedoch nicht spezifiziert.

Der Verbundalgorithmus von BSC erzeugt die Werte eines Zustandes in umgekehrter zeitlicher Reihenfolge. Damit kann vermieden werden, dass in Zuständen Tupel erzeugt werden, die nie zu einem Ergebnis beitragen, weil sie zu alt sind und mit dem nächsten neuen Wert verworfen werden. Falls dieser Fall eintritt, bricht der Algorithmus für den aktuellen Zustand ab und bearbeitet den nächsten Zustand. Die Migration mit BSC ist beendet, wenn alle Zwischenzustände vollständig berechnet sind. Die Migrationsdauer ist dann $T_M \leq w$. Im Migrationsbeispiel würde zunächst $\mathcal{Z}_{BC}$ errechnet und anschließend $\mathcal{Z}_{BCD}$.

HybMig besitzt im Vergleich zu MS und PT einige wesentliche Verbesserungen. Während der Migration werden kontinuierlich Ergebnisse erzeugt. Im Unterschied zu PT bleibt die zeitliche Ordnung erhalten, auch mit BSC. Zudem treten keine Verringerung des Durchsatzes am Ausgangsdatenstrom und keine plötzliche Freigabe von Datentupeln des neuen Plans auf. Der Durchsatz während der Migration ist damit gleichbleibend und ähnlich wie im Normalbetrieb, wenn keine Migration durchgeführt wird. Die Migrationsdauer entspricht zwar der von PT, kann aber durch die Verwendung von BSC unter Umständen reduziert werden. Trotzdem können nachfolgende Probleme für HybMig festgehalten werden.

(a) In der beschriebenen Form ist HybMig ausschließlich auf Bäume von Verbundoperatoren mit zeitbasierten gleitenden Fenstern und bei Verwendung einer globalen Fenstergröße anwendbar. Die Migration von Anfragen mit anderen zustandsbehafteten Operatoren oder anderen Fenstertypen ist nicht möglich.

(b) HybMig lässt sich nur in zentralisierten Systemen verwenden, da gemeinsame Zustände benutzt werden. In verteilten Systemen müssen Zustände zwischen Knoten transferiert werden, um für Operatoren zur Verfügung zu stehen. Dieser Prozess benötigt Zeit und ist in HybMig nicht vorgesehen.

(c) Zu Beginn der Migration wird der alte Plan kurz unterbrochen, um Ein- und Ausgangsströme mit beiden Boxen zu verbinden und den Zugriff der neuen Operatoren auf gemeinsame Zustände einzurichten. Die Unterbrechung kann zu einer Verzögerung der Verarbeitung und von Ergebnissen führen.

(d) Da bei gemeinsamen Zuständen mehrere Operatoren auf den gleichen Zustand zugreifen, ist ein Mechanismus notwendig, damit empfangene Datenstromwerte nicht

mehrfach in diesen Zustand eingefügt werden. Die Realisierung von HybMig ist nicht beschrieben. Es kann aber davon ausgegangen werden, dass besondere Methoden und ein zusätzlicher zeitlicher Aufwand notwendig sind.

(e) Um eine gefilterte Ausgabe der alten Box zu erhalten, ist die Verwendung einer Merkmalstabelle und spezieller Algorithmen notwendig. Eine automatische Erstellung der Merkmalstabelle ist zwar vorstellbar, aber nicht beschrieben. In komplexen Anfragen könnte das Erstellen der Tabelle schwierig sein.

(f) Der Einsatz spezieller Verbundalgorithmen während der Migration stellt eine Modifikation der Originalanfrage zur Laufzeit dar.

(g) Ohne die Verwendung von BSC beträgt die Migrationsdauer w, da die leeren Zustände kontinuierlich mit Werten gefüllt werden müssen. Wie bei PT ist dies insbesondere bei großen Fenstern kritisch, da eine lange Migrationsdauer auftritt.

(h) HybMig wird wie alle bisher vorgestellten Migrationsstrategien ausschließlich auf semantisch äquivalente Pläne angewendet.

Die größte Schwierigkeit von HybMig ist die Einschränkung auf Bäume von Verbundoperatoren und zeitbasierte gleitende Fenster. Um beliebige Operatoren zu unterstützen wurde in [YKPS07] die Migrationsstrategie GHM (Generalized Hybrid Migration) vorgeschlagen. Die Beschränkung auf zeitbasierte gleitende Fenster bleibt jedoch bestehen. GHM kombiniert Funktionen von GMS und GPT. Alter und neuer Plan werden ebenso jeweils als Black Box behandelt.

GHM ist vom Aufbau GPT ähnlich, eingehende Datenströme werden aber nicht direkt mit dem neuen Plan verbunden, sondern durch FIFO-Warteschlangen. Wie bei GMS werden Werte aus den Zuständen des alten Plans extrahiert, um den neuen Plan zu initialisieren. Die extrahierten Werte werden in ihrer zeitlichen Reihenfolge in die Warteschlangen eingefügt. Hinzu kommen die neuen Werte der Eingangsdatenströme. Während der Migration ist der alte Plan für das Erzeugen der Ergebnisse zuständig. Deshalb arbeitet er mit einer höheren Priorität. Die Initialisierung des neuen Plans erfolgt mit einer niedrigeren Priorität während der Pausenzeiten des Systems. Alle Ergebnisse, die während der Migration von der neuen Box erzeugt werden, werden wie bei GPT von einer temporären Datensenke empfangen und direkt verworfen. Die Migration endet, wenn alle Eingangswarteschlangen zum ersten Mal gleichzeitig leer sind. Dann werden die Ein- und Ausgangsdatenströme direkt mit dem neuen Plan verbunden und der alte Plan, die Warteschlangen sowie die temporäre Datensenke werden verworfen. Die Migrationsdauer von GHM beträgt maximal w. Bei ausreichend Pausenzeiten des Systems kann die Migration jedoch deutlich schneller abgeschlossen werden.

Mit GHM können die meisten Probleme von HybMig vermieden werden. Die größte Verbesserung stellt die Verwendung von beliebigen Operatoren mit zeitbasierten gleitenden Fenstern dar. Im Prinzip sollte GHM auch für andere Fenstertypen anwendbar

sein, dafür sind aber unter Umständen zusätzliche Verfahren zur Bestimmung des Migrationsendes notwendig. Außerdem ist GHM durch den Verzicht auf gemeinsame Warteschlangen nicht nur zentralisiert, sondern auch verteilt anwendbar. Da das Verfahren von GHM sich wesentlich von HybMig unterscheidet, sind die Probleme von GHM separat aufgelistet.

(a) Wie BSC bei HybMig läuft die Initialisierung des neuen Plans als Hintergrundprozess. Nicht klar ist, wie die Unterbrechung der Initialisierung durch den Hauptprozess gehandhabt wird. Beim Eintreffen eines neuen Tupels wird dieses zwar priorisiert verarbeitet und der Hintergrundprozess unterbrochen, der genaue Ablauf ist aber nicht beschrieben.

(b) Die Zustandsextraktion und der anschließende Transfer benötigen Zeit. Durch die Realisierung mit FIFO-Warteschlangen muss der Zustandstransfer jedoch bis zum Eintreffen des nächsten Eingangswertes abgeschlossen sein. Dies ist besonders bei aperiodischen Datenströmen nicht sicher zu bewerkstelligen. Wird diese Bedingung schließlich nicht erfüllt, werden während der Migration alte Zustandswerte und neue Eingangsdatenstromwerte gemischt in die Warteschlangen eingefügt. Deshalb ist die Verwendung von FIFO-Warteschlangen nicht geeignet. Eine Alternative wäre die Verwendung von sortierenden Warteschlangen.

(c) Im Unterschied zu HybMig wird bei GHM der gesamte Zustand übertragen. Dies führt dazu, dass Werte übertragen werden, die nicht zu Ergebnissen beitragen. Die Übertragung von ungenutzten Datentupeln verlängert die Migration.

(d) Das Verarbeiten der Werte aus den Warteschlangen ist notwendig, um die Zustände der neuen Box zu füllen. Dabei wird für jeden Wert CPU-Zeit verbraucht. Dies fällt vor allem bei zeitintensiven Operationen ins Gewicht und verlangsamt die Migration. Existieren Zustandspaare, die durch Transfer schneller gefüllt werden können als durch Berechnung, kann durch direkten Zustandstransfer die Migration verkürzt werden. Dies ist allerdings bei GHM nicht vorgesehen.

3.3 Andere adaptive Datenstromsysteme

Dieser Abschnitt gibt einen kurzen Überblick über andere Datenstromsysteme, die Anpassungen während der Laufzeit oder Zustandstransfer durchführen, dabei aber die Migration nicht als Schwerpunkt behandeln und deshalb auch keine Migrationsstrategien vorgestellt haben.

3.3.1 OSIRIS-SE

OSIRIS-SE (Open Service Infrastructure for Reliable and Integrated Process Support – Stream Enabled [BSS06, Bre08]) ist ein DSMS, welches die Verteilung von Operatoren

auf heterogene Verarbeitungsknoten unterstützt. Das primäre Einsatzfeld sind Anwendungen im Gesundheitswesen zur Patientenüberwachung, weshalb die Zuverlässigkeit des Systems im Mittelpunkt der Forschung steht. OSIRIS-SE implementiert Konzepte, die die Zuverlässigkeit von Operatoren innerhalb definierter Randbedingungen garantieren. Ein wesentlicher Bestandteil ist die zustandserhaltende Operatormigration zur Laufzeit.

Die Operatormigration wird ausgelöst, wenn Systemfehler, z. B. Fehler von Verarbeitungsknoten, Netzwerkfehler oder Fehler von Operatoren, die Ausführung einer Anfrage unterbrechen. Zur Identifikation von Fehlern wird eine Wartezeit definiert. Sendet ein Operator für die Dauer der Wartezeit kein Ergebnis, gilt er als fehlerhaft und wird migriert. Betroffene Operatoren werden dann auf einem Sicherungsknoten wiederhergestellt. Dafür werden die Zustände der Operatoren unter Verwendung von zuvor erstellten Sicherungspunkten initialisiert. Danach kann die Bearbeitung fortgesetzt werden.

Ein Sicherungspunkt beinhaltet neben dem inneren Zustand des Operators auch den Zustand der Ausgangswarteschlange, die Routing-Informationen mit welchen anderen Systemelementen der Operator verbunden ist und den Zeitkontext, der angibt, welches Datenstromelement zuletzt empfangen wurde und welches zuletzt versendet wurde.

Sicherungspunkte werden regelmäßig während der gesamten Laufzeit angelegt und auf einem Knoten im Netzwerk gespeichert. Jeder Operator einer Anfrage erzeugt seine eigenen Sicherungspunkte. Für das Erstellen von Sicherungspunkten („Operator Checkpointing“) wurden zwei Methoden vorgestellt – koordinierte und unkoordinierte Sicherungspunkte.

Bei unkoordinierten Sicherungspunkten erstellt jeder Operator seine Sicherungspunkte ohne Berücksichtigung benachbarter Operatoren. Dafür werden alle Bestandteile des Zustandes auf einem anderen Knoten abgespeichert. Dies erzeugt eine permanente Mehrbelastung für CPU, Speicher und Netzwerk, die besonders für Speicher und Netzwerk größer ist als der Bedarf für die Datenstromverarbeitung selbst.

Für koordinierte Sicherungspunkte ist eine Koordination zwischen benachbarten Knoten notwendig. Operatoren senden in Richtung Datensenke eine Sicherungsanfrage bis der letzte Operator erreicht ist. In umgekehrter Richtung werden die Sicherungspunkte erzeugt. Jeder Operator benachrichtigt nach Abschluss der Sicherungsprozedur seine Vorgänger. Durch die Koordination zwischen den Knoten kann die Mehrbelastung reduziert werden.

Eine weitere Funktion von OSIRIS-SE unterstützt die Verteilung der Operatoren auf heterogene Knoten. Muss ein Operator auf einen Knoten mit geringerer Leistungsfähigkeit verschoben werden, ist es möglich, einen weniger genauen Algorithmus zu nutzen, sofern dieser zuvor spezifiziert wurde. Das ermöglicht, weniger Ressourcen zu verbrauchen und eine Kompatibilität des Algorithmus mit dem neuen ausführenden Knoten zu erreichen.

Migration in OSIRIS-SE wird ausschließlich als Operatormigration verstanden, nicht als Anfragenmigration. Das heißt, wenn eine Migration notwendig ist, werden die betroffenen Operatoren separat migriert. Fehlerhafte Knoten werden durch neue ersetzt, und die ursprüngliche Anfrage bleibt erhalten. Das vereinfacht einerseits die Migrationspro-

zesse, da neue Konstellationen, wie neu entstehende Zustände (z. B. $\mathcal{Z}_{BC}$ und $\mathcal{Z}_{BCD}$ im oberen Beispiel), nicht auftreten und somit auch nicht berücksichtigt werden müssen. Zudem gibt es keine Einschränkungen für Operator- oder Fenstertypen, da die inneren Zustände, d. h. sämtliche Fensterinhalte, aus den Sicherungspunkten wiederhergestellt werden können. Andererseits sind die Anwendungsmöglichkeiten dadurch eingeschränkt, beispielsweise lassen sich mit einer Migration keine Kompositionsänderungen wie in Abbildung 3.1 umsetzen. OSIRIS-SE ist jedoch für solche Aufgaben nicht entwickelt worden, sondern nutzt die Migration als Mittel zur Erhöhung der Zuverlässigkeit eines verteilten DSMS indem fehlerhafte Operatoren auf fehlerfreien Knoten neu gestartet werden.

Eine Migrationsdauer ist in OSIRIS-SE nicht definiert. Stattdessen wird eine maximale Zeit festgelegt, wie lange eine Anfrage ohne Ergebniswert eines Operators auskommt. In diesem Zeitraum muss bei Bedarf ein defekter Operator zunächst erkannt und anschließend aus dem letzten Sicherungspunkt vollständig wiederhergestellt werden. Die Migrationsdauer entspricht somit mindestens der Wiederherstellungsdauer. Nach der Migration ist es möglich, dass bereits bekannte Ergebnisse (Duplikate) produziert werden. Zwar werden in OSIRIS-SE Duplikate ignoriert, erwartete Ergebnisse werden aber dadurch zusätzlich verzögert.

Auch wenn die Migrationsmethoden von OSIRIS-SE nicht zum Zweck der Anfrageoptimierung entwickelt wurden, werden nachfolgend mögliche Probleme genannt, würden die Methoden zu diesem Zweck eingesetzt. Durch die regelmäßige Erstellung von Sicherungspunkten entsteht eine Mehrbelastung für CPU und Netzwerk. Das Abspeichern der Sicherungspunkte führt mindestens zu einer Verdopplung des Speicherbedarfs, da der gesamte innere Zustand als auch die Ausgangswarteschlange im Sicherungspunkt abgelegt werden. Die Migrationsdauer entspricht mindestens der Wiederherstellungsdauer. Hinzu kommt die Verzögerung durch Duplikate. Während der Wiederherstellung eines Operators und auch während der Fehlererkennung werden keine Ergebnisse ausgegeben.

3.3.2 StreamMine

Ähnliche Konzepte wie in OSIRIS-SE werden im System StreamMine [BFF09a] genutzt, um Anfragen in einem verteilten Datenstromsystem nach einem ungeplanten Abbruch wiederherzustellen. Zum Erreichen einer Fehlertoleranz werden Operatoren parallelisiert ausgeführt. Nach einem Abbruch ist das Ziel, Operatoren exakt wiederherzustellen, d. h. die produzierten Ergebnisse nach der Wiederherstellung sind so, als wäre nie ein Abbruch aufgetreten. Dies erfordert die korrekte Wiederherstellung der Zustände sowie der zeitlichen Reihenfolge der Datentupel. Besonders nichtdeterministische, zustandsbehaftete Operatoren erfordern besondere Mechanismen.

Um einen Zustand wiederherstellen zu können, werden fortlaufend die Eingangswerte und alle nichtdeterministischen Aktionen aufgezeichnet. Durch zusätzliche Sicherungspunkte kann die Menge der aufgezeichneten Daten reduziert werden, indem Datensätze, die älter sind als der letzte Sicherungspunkt, verworfen werden. Mit Hilfe der Sicherungspunkte und der aufgezeichneten Eingangswerte und Aktionen kann der Zustand

der Anfrage zum Zeitpunkt des Absturzes exakt rekonstruiert werden.

Durch die Wiederherstellung entsteht eine temporäre Verzögerung von Ergebnissen. Es ist aber möglich, „spekulative Werte“ auszugeben. Man unterscheidet zwischen endgültigen und spekulativen Ergebnissen und markiert diese entsprechend. Spekulative Ergebnisse können zu einem späteren Zeitpunkt korrigiert werden. Besonders während der Wiederanlaufphase sorgen spekulative Ergebnisse dafür, dass nachfolgende Operatoren weiterarbeiten können, wenngleich die Umwandlung zu endgültigen Ergebnissen während des Wiederanlaufs länger dauert, als im Normalbetrieb. Auch StreamMine ist in der Lage, mit Duplikaten umzugehen und diese zu ignorieren.

In [BFF09b] wird eine Prozedur zur Wiederherstellung von fehlerhaften Replikaten vorgestellt. Als einer von drei Wiederherstellungsschritten, wird der Zustand von fehlerfreien Replikaten auf das neu gestartete Replikat übertragen. Eine detaillierte Beschreibung des Zustandstransfers gibt es allerdings nicht.

3.3.3 Aurora, Medusa und Borealis

Medusa und Borealis sind verteilte Datenstromsysteme, für die Methoden zur Laufzeitanpassung von Datenstromanfragen und Zustandstransfer beschrieben wurden. Beide Systeme basieren auf Aurora [Aur09, BBC+04], einem DSMS der ersten Generation. Bereits in Aurora, einem monolithischen DSMS, konnten zum Zweck der Anfrageoptimierung Kompositionsänderungen durchgeführt werden.

Für die Anfragenanpassung in Aurora sind Verbindungspunkte („Connection Points“) notwendig, die bei Bedarf durch einen Benutzer zwischen Operatoren eingebunden werden können und die alle Werte des Datenstroms zwischen diesen Operatoren aufzeichnen und für eine spezifizierte Zeit speichern. Durch das Einfügen von Verbindungspunkten wird die gesamte Anfrage in Teilanfragen gegliedert. Diese Teilanfragen können automatisch optimiert werden.

Ausgehend von einer nicht-optimierten Anfrage werden Laufzeitstatistiken erfasst, beispielsweise die durchschnittlichen Ausführungskosten. Mit Hilfe dieser statistischen Werte wird versucht, die Teilanfragen zu optimieren. Dieser Schritt geschieht parallel zur laufenden Anfrage. Zu den Optimierungsstrategien [ACÇ+03] zählen Einfügen von Projektion-Operatoren, Kombinieren von Operatoren oder Umsortieren von Operatoren. Alle Modifikationen stellen Kompositionsänderungen dar. Die Kombination von Operatoren bezieht sich hauptsächlich auf zustandsfreie Operatoren, z. B. Map oder Filter, um das Datenaufkommen in der Anfrage frühzeitig zu reduzieren.

Alle optimierten Teilanfragen werden danach im Aurora-System erstellt. Mit den in Verbindungspunkten gespeicherten Eingangswerten ist es möglich, die neuen Teilanfragen zu initialisieren. Zur Durchführung der Optimierung werden die ursprünglichen Teilanfragen angehalten. Alle Verbindungspunkte auf eingehenden Datenströmen der Teilanfrage puffern ankommende Tupel für den Zeitraum der Optimierung. Die Ausgangsdatenströme dieser Teilanfragen zeigen für den Optimierungszeitraum keine Ergebnisse.

Die Möglichkeit, Anfragen zu optimieren, besteht ausschließlich zwischen Verbindungspunkten, d. h. für Teilanfragen. Eine Optimierung über Verbindungspunkte hinweg ist nicht möglich. Die Optimierung erfordert eine hinreichende Menge an Daten in den Verbindungspunkten, d. h. Verbindungspunkte müssen zeitig genug angelegt werden, damit alle zustandsbehafteten Operatoren der Teilanfrage initialisiert werden können. Zudem ist zusätzlicher Speicher notwendig, um die in den Verbindungspunkten erfassten Datenstromtupel abzuspeichern. In Aurora wird ein zentraler Speichermanager genutzt, der die Daten aller Verbindungspunkte verwaltet. Außerdem bietet er die Möglichkeit, gemeinsame Warteschlangen zu nutzen, welche gleichzeitig die Zustände der Operatoren darstellen.

Insgesamt erlaubt Aurora Kompositionsänderungen während der Laufzeit zum Zweck der Anfrageoptimierung. Die Notwendigkeit von Verbindungspunkten und die damit verbundenen Anforderungen an Speicher und zeitliche Voraussicht zur Einrichtung von Verbindungspunkten schränken die Anpassung jedoch ein. Änderungen von Algorithmen oder von Parametern scheinen nicht vorgesehen zu sein.

Medusa[8] setzt die Konzepte des Aurora-Systems in einer verteilten Umgebung um. Ziele sind Lastverteilung und Fehlertoleranz. In Medusa ist es möglich, Operatoren von überlasteten Knoten zu verschieben. Für zustandsbehaftete Operatoren wird deren innerer Zustand bei der Verschiebung berücksichtigt. Im Vergleich zu Aurora verfügt Medusa daher über zusätzliche Methoden für den Zustandstransfer. Weiterhin gilt, dass Modifikationen, also auch das Verschieben von Operatoren auf andere Bearbeitungsknoten, ausschließlich zwischen Verbindungspunkten möglich ist [CBB+03].

Für den Transfer wird zunächst die entsprechende Teilanfrage angehalten und alle gepufferten Werte in der Teilanfrage werden bearbeitet. Sind alle verarbeitet, wird der zu verschiebende Operator aus der Teilanfrage entfernt und sein innerer Zustand extrahiert. Anschließend werden die Verbindungen zu Ein- oder Ausgangsdatenströmen, z. B. zum neuen Knoten, neu erstellt und danach kann die Verarbeitung der Teilanfrage fortgeführt werden. Schließlich wird der neue Bearbeitungsknoten angehalten und ein gleichartiger Operator angelegt. Dieser wird mit dem transferierten Zustand initialisiert und die Verarbeitung wird fortgesetzt. Damit ist die Operatormigration beendet.

Die Operatormigration ähnelt der Migrationsstrategie MS. Eingeschränkt ist sie, wie in Aurora, durch die notwendigen Verbindungspunkte. Während der Migration pausiert die Verarbeitung, und es werden keine Ergebnisse ausgegeben.

Borealis [Bor09, AAB+05] ist als Nachfolger von Aurora und Medusa ein Datenstromsystem der zweiten Generation und wird erweiterten Anforderungen gerecht, z. B. Verteilung auf heterogene Geräte, Laufzeitanpassungen ohne Beeinflussung des Systems oder Fehlertoleranz.

Für Borealis wurden Laufzeitanpassungen in Form von Parameter- und Kompositionsänderungen beschrieben. Parameteränderungen werden durch spezielle Steuernachrich-

[8] Die gleichen Konzepte werden auch von Aurora* [CBB+03] implementiert, einer weiteren verteilten Anwendung von Aurora.

ten realisiert, die neben den neuen Parametern auch eine Zeitbedingung enthalten, ab wann die neuen Parameter gültig sind. Damit ist es möglich, beispielsweise Fenstergrößen, Abtastraten oder parametrierbare Algorithmen zu modifizieren. Diese Algorithmen sind in einer Algorithmendatenbank persistent gespeichert.

Für die Umsetzung von Kompositionsänderungen wurde in Borealis die Funktionalität der Verbindungspunkte erweitert. Es ist möglich, neue Verzweigungen an Verbindungspunkten anzulegen („Connection Point Views“), um Ergebnisse mit alternativen Teilanfragen zu berechnen. Somit können mehrere Versionen von Teilanfragen parallel existieren, welche zudem unabhängig voneinander modifizierbar sind. Mit Hilfe der Verbindungspunkte werden dafür simulierte Tupel erzeugt oder vergangene Werte wiedergegeben. Die normale Verarbeitung der Anfrage wird dadurch nicht beeinträchtigt, allerdings sind keine Implementierungsdetails beschrieben. Die parallelen Teilanfragen können zur Entwicklung von effektiveren Alternativen und zur Fehlersuche vor der Inbetriebnahme genutzt werden. Im laufenden Betrieb ist es möglich, zwischen Teilanfragen umzuschalten. Diese Möglichkeit wird beispielsweise genutzt, um bei Überlast automatisch zu günstigeren Anfragen oder Teilanfragen zu wechseln.

Weiterhin kann das Umschalten zum Erreichen einer Fehlertoleranz genutzt werden [BBMS08]. Dafür werden Bearbeitungsknoten repliziert. Bei einem Fehler wird versucht, automatisch zu einem stabilen Replikat zu wechseln. Ist kein stabiles Replikat vorhanden, ist es auch möglich, mit Ergebnissen eines potentiell fehlerhaften Replikats weiter zu arbeiten. Erstellte Ergebnisse werden dann aber als „vorläufig“ markiert.

Zur Wiederherstellung wird ein abgestürzter Bearbeitungsknoten neu gestartet, wobei er dadurch mit leeren Zuständen zu arbeiten beginnt. Die Zustände werden jedoch rekonstruiert, wofür zwei Ansätze vorgestellt wurden [BBMS05]. Die bevorzugte Methode nutzt die Möglichkeit, den letzten stabilen Zustand von einem Sicherungspunkt zu laden. Das erfordert jedoch die fortlaufende Speicherung von Sicherungspunkten. In Borealis enthält ein Sicherungspunkt die Daten eines kompletten Bearbeitungsknotens. Die andere Methode annulliert vorläufige Ergebnisse und stellt einen vorherigen Zustand wieder her. Dazu werden vergangene, gepufferte Eingangsdaten verwendet, welche von den Operatoren für eine bestimmt Zeit gespeichert werden.

Durch die Verwendung von vorläufigen Tupeln müssen nach einem Fehler unter Umständen mehrere Knoten ihre Zustände aktualisieren und abgleichen. Das betrifft alle Knoten, die sich in der Bearbeitung hinter einem fehlerhaften Knoten befinden. Für den Zeitraum der Wiederherstellung werden keine neuen Tupel von diesen Knoten verarbeitet. Dies ist in Borealis akzeptabel, solange die Verzögerung unter einer spezifizierten Latenz bleibt.

Borealis implementiert einige Konzepte zur Anfragenmodifikation und Zustandswiederherstellung. Bei der Anfragenmodifikation bleiben weiterhin die Verbindungspunkte die zentralen Elemente. Nur an ihnen sind Verzweigungen möglich. Sie speichern alle Daten für Verzweigungen. Dies erfordert, dass Verbindungspunkte rechtzeitig angelegt werden und hinreichend Speicher zur Verfügung steht, da Verzweigungen den Speicherbedarf vervielfachen. Zudem muss der Speicher permanent verfügbar sein und nicht nur

während der Migration. Zustandstransfer wurde für Borealis nicht beschrieben.

Eine Form der Anfragenmodifikation ist die Anpassung von Algorithmen. Diese müssen parametrierbar sein, um während der Laufzeit geändert zu werden. Die Modifikation von komplexen Algorithmen, z. B. durch Aktualisierung des Quellcodes lassen sich mit Parametrierung nicht lösen. Die Algorithmen sind persistent in einer Algorithmendatenbank gespeichert. Es ist nicht beschrieben, ob diese Datenbank zur Laufzeit aktualisiert werden kann, um Code zu aktualisieren oder neue Algorithmen abzulegen. Algorithmen sind also nur bedingt modifizierbar.

Anfragenmodifikation zur Erhöhung der Fehlertoleranz wird in Borealis durch das Umschalten von Teilanfragen realisiert. Dabei wird nur zwischen gesamten Knoten umgeschaltet, nicht zwischen einzelnen Operatoren. Modifikationen zur Optimierung von Anfragen werden in diesem Zusammenhang nicht diskutiert. Fehlerhafte Knoten werden neu gestartet und ihr Zustand rekonstruiert. Der Zustand wird vorzugsweise mit Daten aus Sicherungspunkten wiederhergestellt. Methoden zum Zustandstransfer wurden auch für diesen Anwendungsfall nicht beschrieben.

3.3.4 STREAM

STREAM [STR09, ABB+04] ist eines der ersten und gleichzeitig bekanntesten Datenstromsysteme. Schwerpunkt von STREAM ist die Berechnung von approximierten Ergebnissen [BBD+02, MWA+03] innerhalb von benutzerdefinierten Toleranzen zugunsten einer Reduktion der Systemlast. Die Optimierung der Anfrageausführung wird beispielsweise durch Filtern von Datenströmen erreicht. Insbesondere bei Datenquellen aus Sensornetzwerken kann dadurch die Anzahl der zu übertragenden Nachrichten minimiert werden. STREAM ist ein zentralisiertes System. Für innere Zustände (in STREAM: „synopses“) wurden Konzepte für gemeinsame Zustände (engl. „synopsis sharing“) entwickelt [ABB+04].

Die vorgestellten Ansätze für verteilte Datenstromverarbeitung [OJW03, BO03] beschränken sich auf verteilte Datenquellen und führen die Verarbeitung im zentralen System durch. Die Datenquellen liegen auf parametrierbaren Verarbeitungsknoten. So werden in [OJW03] die Knoten der verteilten Quellen durch Filter ergänzt. Mit dem Ziel den Durchsatz zu minimieren, verfeinern die Filter ihre Genauigkeit während des Betriebs periodisch und automatisch. Bei Bedarf werden sie durch das Verarbeitungssystem benachrichtigt, die Genauigkeit zu reduzieren, wodurch wieder mehr Werte des Eingangsdatenstroms übertragen werden. In [BO03] ist die Aufgabe der Quellen ausschließlich Ausreißerwerte zu identifizieren und zu übermitteln. Dafür ist über alle verteilten Datenquellen hinweg die Kenntnis der Datenstrom- und Ausreißercharakteristik notwendig. Deshalb müssen nach Auftreten eines Ausreißers die Datenquellenknoten mit den aktuellen Einstellungen aktualisiert werden. Dies wird durch eine Parameterübertragung an die betreffenden Verarbeitungsknoten realisiert.

Beide Ansätze haben die verteilten Datenquellenknoten gemeinsam, welche durch Aktualisierung von Parametern während der Laufzeit an aktuelle Gegebenheiten angepasst

werden können. Abgesehen von der Filterung wird keine verteilte Verarbeitung durchgeführt. Stattdessen werden die übertragenen Daten im zentralisierten Datenstromsystem verarbeitet. Operatormigration und Zustandstransfer wurden für STREAM nicht beschrieben.

Mit StreaMon [BW04] wurde eine Datenstrom-Infrastruktur für STREAM entwickelt, welche Anfragen zur Laufzeit optimieren kann. Dafür wird das System um ein Beobachtungs- und Analysemodul („Profiler“) und ein Optimierungsmodul („Reoptimizer“) ergänzt. Der Profiler überwacht die Datenströme und erzeugt Statistiken, die vom Reoptimizer für eine Bewertung herangezogen werden können. Wird durch den Reoptimizer eine Verbesserungsmöglichkeit erkannt, erfolgt eine Anpassung der Anfrage zur Laufzeit. Es können durch Parametrierung die Größen der inneren Zustände gesteuert werden, um den Speicherverbrauch zu minimieren [BSW04]. Außerdem besteht die Möglichkeit, die Verarbeitungsreihenfolge von zustandslosen Verbundoperatoren neu zu sortieren [BMM+04], d. h. eine Kompositionsänderung durchzuführen. Eine Erweiterung dazu stellt einen Mechanismus zur Verfügung, der es erlaubt, Zwischenergebnisse bei Bedarf in Zuständen zu speichern [BMWM05], wodurch die Verarbeitungsreihenfolge von zustandsbehafteten Verbundoperatoren optimiert werden kann. STREAM wird auch mit StreaMon zentralisiert ausgeführt. Die Änderungen werden direkt an den Anfragepläne durchgeführt, wodurch Ausgabeverzögerungen hervorgerufen werden können, insbesondere bei der Anpassung der zustandsbehafteten Operatoren. Operatormigration und Zustandstransfer wurden im Zusammenhang mit StreaMon nicht diskutiert.

3.3.5 TelegraphCQ

TelegraphCQ [Tel09, CCD+03] ist wie Aurora und STREAM ein Datenstromsystem der ersten Generation. Es unterscheidet sich von diesen Systemen jedoch, da es zur Ausführung keinen festen Anfragen verwendet, sondern die Verarbeitungsreihenfolge der Operatoren während der Laufzeit gemäß den aktuellen Systemeigenschaften und pro Tupel anpasst. Zustandserhaltung und -migration werden in diesem Zusammenhang als Herausforderung genannt. Die zugrunde liegenden Konzepte Eddies, SteMs und Flux werden nachfolgend im Einzelnen erläutert.

Eddies[AH00] wurden bereits vor TelegraphCQ als Konzept beschrieben, bilden jedoch eine Kerntechnologie für dieses Datenstromsystem. Eddies sind die Kommunikationsmodule des dynamisch anpassbaren DSMS und sind für das individuelle Steuern (engl. „routing“) der Tupel zuständig. Dafür besitzt jedes Tupel einen eigenen Zustand, welcher die notwendigen und absolvierten Arbeitsschritte dokumentiert.

SteMs (State Modules) [RDH03] sind eigenständige Zustandsmodule in TelegraphCQ. Neben der Möglichkeit zustandsbehaftete Operatoren als komplettes Modul zu realisieren, können Verbundoperatoren in TelegraphCQ mit Hilfe von SteMs modelliert werden. Der innere Zustand eines Verbundoperators kann somit getrennt von der Logik durch einen Eddy kontrolliert werden. Ein SteM ist im Prinzip ein halber Verbundoperator und unterstützt die entsprechenden Zustandsoperationen, d. h. Einfügen (engl. „insert“)

und Entfernen (engl. „evict“) von Tupeln sowie die Ermittlung eines Ergebnisses (engl. „probe“). Der Einsatz von SteMs ermöglicht die Verwendung von gemeinsamen Zuständen, erhöht aber auch die Möglichkeit von Fehlern, z. B. doppelte oder fehlende Werte oder Endlosschleifen. Deshalb wurden Beschränkungen definiert, die ein Eddy bei der Steuerung zu befolgen hat.

Das Konzept, Verbundoperatoren in SteMs und Logik zu zerlegen, ermöglicht eine bessere Optimierung durch Eddies. Jedoch wurden SteMs lediglich für Tupel von Eingangsdatenströmen aber nicht für zusammengesetzte Tupel beschrieben. Außerdem scheint die Verwendung auf Verbundoperatoren beschränkt, andere zustandsbehaftete Operatoren wurden nicht diskutiert. Eine Benutzung in verteilten Umgebungen erscheint realisierbar, wurde aber in [RDH03] nicht betrachtet.

Flux (Fault-tolerant Load-balancing eXchange) [SHCF02] ist ein allgemeines Konzept für die Parallelisierung von Teilanfragen. Es wurde in TelegraphCQ in Kombination mit Eddies implementiert, ist aber im Prinzip unabhängig vom Datenstromsystem. Laut [SHCF02] ist auch eine Implementierung in Aurora möglich, und auch [Krä07] zieht eine Verwendung in PIPES in Betracht. Flux kann für Lastverteilung [SHCF02] oder für Fehlertoleranz [SHB04] genutzt werden. Dafür werden zusätzliche Operatoren in eine Anfrage eingefügt, die zunächst einen Datenstrom aufteilen und auf die parallelen Teilanfragen verteilen und schließlich wieder zusammenführen. Zum Teil sind die zusätzlichen Operatoren mit den Verbindungspunkten von Aurora vergleichbar. Der wesentliche Unterschied besteht jedoch darin, dass innere Zustände nicht in den Verbindungspunkten gespeichert werden, sondern in den Replikaten der Teilanfragen.

Eine Lastverteilung wird durch das Verschieben von Operatorpartitionen auf andere Verarbeitungsknoten realisiert. Die Operatorpartitionen entsprechen Teilen des inneren Zustandes, wobei es viel mehr Partitionen als Verarbeitungsknoten gibt. Fehlertoleranz wird durch das Ausführen von redundanten Teilanfragen auf verschiedenen Verarbeitungsknoten erreicht. Fällt eine Anfrage durch einen Fehler aus, kann ein Replikat übernehmen und die ausgefallene Teilanfrage wird neu gestartet. Für die Initialisierung wird der Zustand des laufenden Replikats integriert. Eine notwendige Funktion, sowohl für die Verschiebung der Operatorpartitionen als auch für die Initialisierung nach einem Ausfall, ist die Übertragung des Zustands auf einen anderen Verarbeitungsknoten. Dafür wird ein Zustandstransferprotokoll („state movement protocol“ [SHCF02]) verwendet. Mit Hilfe des Protokolls werden folgende Schritte ausgeführt. Der zusätzliche Operator an der Eingangsseite der Teilanfrage pausiert den Datenfluss. Danach wird der Zustand bzw. die Partition extrahiert und an den neuen Verarbeitungsknoten geschickt. Dort wird er in die Datenstruktur integriert und nach einer Synchronisierung kann die Bearbeitung fortgesetzt werden. Der Nachteil dieser Vorgehensweise ist die Unterbrechung der laufenden Anfrage für die Verlagerung oder die Wiederherstellung des Zustandes. Anfragen, die nicht von einem Transfer betroffen sind, können aber fortgeführt werden. Durch die zahlreichen Benachrichtigungen und Bestätigungen der teilnehmenden Operatoren kommt es, auch im Betrieb ohne Zustandstransfer, zu einem Mehraufwand bei der Kommunikation. Zudem entstehen durch das Verwenden von zusätzlichen Opera-

toren ähnliche Nachteile, wie bei Auroras Verbindungspunkten. Zwar wurde Flux nicht für Anfragenanpassungen oder eine Operatormigration zur Laufzeit gezeigt, sondern für dynamische Optimierung durch Lastverteilung und Fehlertoleranz, eine Verwendung des Zustandstransferprotokolls für die Migration könnte aber sinnvoll sein. Eine Umsetzung ähnlich MS in CAPE erscheint realisierbar.

3.3.6 NexusDS

NexusDS [CEB+09, CNG+09] ist eine flexible, erweiterbare Middleware zur Steuerung von kontinuierlichen Datenströmen in verteilten, heterogenen Umgebungen. Die Middleware repräsentiert eine Abstraktionsschicht zwischen den physischen Verarbeitungsknoten der kontinuierlichen Anfragen, z. B. ein Sensornetzwerk, und domänenspezifischen Anwendungen. NexusDS stellt Dienste für diese Anwendungen zur Verfügung, beispielsweise Basisdienste, wie Operatoren, oder Domänendienste, z. B. zur Visualisierung oder zur Historisierung von Daten. Zu den Hauptfunktionen des Systems gehören die Integration von benutzerdefinierten Operatoren sowie die Anpassung von Anfragen zur Laufzeit.

Anwendungen übermitteln logische Anfragepläne an die Middleware. Diese werden zunächst optimiert und anschließend unter Berücksichtigung der Eigenschaften der Verarbeitungsknoten fragmentiert und entsprechend verteilt, d. h. eine logische Anfrage wird in eine physische Anfrage umgewandelt und instanziiert. Damit die Fragmentierung der Anfragen realisiert werden kann, sind einerseits statistische Laufzeitinformationen der Verarbeitungsknoten notwendig, z. B. CPU-, Speicher- und Netzwerkauslastung. Hinzu kommen die funktionalen Eigenschaften von Verarbeitungsknoten, etwa ob eine Grafikkarte vorhanden ist. Andererseits beschreiben Operatoren durch Metadaten ihre Anforderungen [CLM10], beispielsweise wie viele Ein- und Ausgänge sie besitzen und welche Datenformate diese unterstützen. Ebenso werden Ausführungsanforderungen angegeben, z. B. welche Hardware für die Verarbeitung erforderlich ist. Ein physischer Operator setzt sich aus Eingangswarteschlangen, einem Input-Manager und der Operatorlogik[9] zusammen. Die Operatorlogik enthält nicht nur den Algorithmus zur Berechnung der Ergebnisse, sondern auch Funktionen zur Erfassung der Laufzeitstatistiken. Input-Manager sind das aktive Kernelement des Operators und bilden die Verbindung zwischen den Eingangswarteschlangen und der Operatorlogik. Operatoren werden je nach Spezialisierung unterteilt in Basisoperatoren, domänenspezifische Operatoren und anwendungsspezifische Operatoren. Ihre Implementierungen werden in einem zentralen Verzeichnis verwaltet und bei Bedarf durch die Verarbeitungsknoten geladen. Neben benutzerdefinierten Operatoren und Laufzeitanpassung wurden die Komprimierung von Datenströmen [CNG+09], Lastabwurf (load shedding) und Scheduling-Strategien [CEB+09, CNG+09] als wesentliche Funktionen des Datenstromsystems genannt.

Da sich Infrastruktur und Anforderungen zur Laufzeit ändern können, sind mit Ne-

[9] In [CEB+09, CNG+09] ebenfalls als „Operator“ bezeichnet.

xusDS Laufzeitanpassungen durchführbar. Zur Unterstützung des Anpassungsprozesses wurden verschiedene Konzepte beschrieben. Zur Durchführung von „leichtgewichtigen Änderungen“ [CEB+09] können alle Bestandteile der Operatoren, d. h. Warteschlangen, Input-Manager und Operatorlogik, parametrisiert werden, falls sie entsprechend implementiert wurden. Damit können beispielsweise Scheduling und Lastabwurf durch Warteschlangen und Input-Manager realisiert werden, um den Durchsatz in Lastsituationen zu verringern. Auch die Ausführungsreihenfolge von Operatoren innerhalb eines Fragments lässt sich damit beeinflussen. All diese lokalen Änderungsmöglichkeiten können während der Laufzeit durchgeführt werden [CEB+09, CNG+09].

Sind die lokalen Änderungen nicht ausreichend, kann ein Anfrageplan zur Laufzeit logisch optimiert und fragmentiert werden. Dafür werden die gleichen Funktionen genutzt, die vor der Ausführung der Anfrage für deren Optimierung verwendet werden. Für beide können Optimierungsstrategien bzw. Fragmentierungsstrategien zur Laufzeit nachgeladen werden. Die Anfrageoptimierung während der Laufzeit kann periodisch oder nach Bedarf, z. B. bei Überlast, ausgelöst werden [CEB+09]. Als unterstützende Funktion besteht die Möglichkeit, Operatoren im laufenden System zu installieren und zu starten. Dieser Mechanismus kann dazu genutzt werden, die Verarbeitungslogik der Operatoren anzupassen. Ebenso können überflüssige Operatoren gestoppt und deinstalliert werden.

Die Realisierung der Anfrageoptimierung wird laut [CEB+09] durch eine Migration erreicht, wobei auf die Systeme CAPE [ZRH04] und PIPES [YKPS07] verwiesen wird. Details zur Migration sind jedoch nicht beschrieben. Eine Betrachtung von zustandsbehafteten Operatoren und vor allem deren Realisierung im System fehlt gänzlich. Deshalb ist nicht klar, wie im System NexusDS die Anfrageoptimierung zur Laufzeit durchgeführt wird und wie die Herausforderungen mit zustandsbehafteten Operatoren gelöst werden.

3.3.7 ACDS

Das System ACDS (Adapting Computational Data Streams [IS00a, IS00b]) ist ein verteilt anwendbares Datenstromsystem für heterogene Umgebungen, welches auf einer Ereignis-basierten Infrastruktur aufbaut und automatische Laufzeitanpassungen unterstützt.

Der allgemeine Anpassungsmechanismus ist ein 2-Phasen-Transaktionsprotokoll. Im ersten Schritt werden alle beteiligten Systemelemente über eine geplante Anpassung benachrichtigt, und es wird auf deren Bestätigung gewartet. Anschließend wird die Anpassung beginnend bei den Quellen in Richtung Datensenke durchgeführt.

Die Anpassungen dienen zur dynamischen Lastverteilung in Abhängigkeit von der Ressourcenverfügbarkeit oder können durch Benutzerinteraktionen ausgelöst werden. Zu den beschriebenen Änderungsmöglichkeiten zählen Parameteranpassungen, Migration und Parallelisierung. Eine Funktion zur Algorithmusanpassung wurde nicht beschrieben. Migration bezeichnet das Verschieben von Operatoren auf andere Verarbeitungsknoten. Logisch wird die Anfrage dabei nicht verändert, d. h. es findet keine Kompositionsänderung statt. Die Migration wird ebenfalls für das Parallelisieren oder Zusammenfassen

(„split“, „merge“) von Operatoren verwendet, wofür in [IS00a] verschiedene Ansätze präsentiert wurden. Durch Parallelisieren können duplizierte Ausgaben entstehen. Für das Zusammenfassen von Operatoren wird das Umkonfigurieren von nachfolgenden Systemelementen als größte Herausforderung genannt. Unklar ist, wie Zustände behandelt oder zusammengeführt werden.

Als Herausforderungen bei der Operatormigration werden die Anpassungen der Verbindungen („rerouting“) und ein konsistenter Zustandstransfer identifiziert. Für den Zustandstransfer bieten Operatoren Methoden zum Speichern und Wiederherstellen der Zustände an. Dies lässt auf einen vollständigen Zustandstransfer schließen, ähnlich MS in CAPE. Eine detaillierte Beschreibung der Durchführung des Zustandstransfers ist allerdings nicht vorhanden. Ebenso fehlt eine Definition, welche Bestandteile zu einem Zustand gehören.

3.4 Überblick über existierende Migrationsstrategien

In Tabelle 3.2 sind die Eigenschaften aller analysierten Datenstromsysteme zusammengefasst. Die erste Gruppe umfasst alle bekannten Migrationsstrategien. Die zweite Gruppe enthält die Systeme, die keine Migrationsstrategien beschreiben, aber Laufzeitanpassung oder Zustandstransfer unterstützen und deren Funktionen für Migrationsstrategien hilfreich sein können. Zum Vergleich sind die Eigenschaften einer hypothetischen, idealen Migrationsstrategie angegeben. Eine Legende der verwendeten Symbole befindet sich unter der Tabelle. Eigenschaften, die nicht zutreffen oder nicht bestimmt werden konnten, wurden offen gelassen.

Für jede Strategie ist das umgesetzte Grundprinzip angegeben – Parallelmethode oder Zustandstransfer. Falls der Zustand übertragen wird, ist auch angegeben, ob der Zustand vollständig oder nur ein Teil übertragen wird. Die Kategorien Ausgabeverzögerung und Korrekte Reihenfolge charakterisieren den Ausgangsdatenstrom während der Migration. Zudem wird spezifiziert, ob eine Migrationsstrategie für beliebige Operatoren und Fenstertypen anwendbar ist und welche Änderungen durchgeführt werden können – Kompositionsänderungen, Algorithmenänderungen und Parameteränderungen. Schließlich werden Einschätzungen aufgeführt, ob eine Strategie in einer verteilten Umgebung angewendet werden kann und wie lange die Migration dauert.

Ideale Migrationsstrategie Die hypothetische, ideale Migrationsstrategie dient zum Vergleich mit den anderen Strategien. Welche Methoden für die Umsetzung einer idealen Migrationsstrategie genutzt werden, ist nicht relevant, sondern ihre Eigenschaften und die zu beobachtenden Ergebnisse. Mit einer idealen Migrationsstrategie sind alle Änderungen in Anfragen mit beliebigen Operatoren und Fenstertypen möglich. Die Ergebnistupel im Ausgangsstrom werden während und nach der Migration in ihrer korrekten Reihenfolge und ohne Verzögerung ausgegeben. Eine ideale Migrationsstrategie ist in verteilten Systemen anwendbar und endet so schnell wie möglich. Falls für die

Strategie	Parallelmethode	Zustandstransfer	Vollständiger Transfer	Ausgabeverzögerung	Korrekte Reihenfolge	Beliebige Operatoren	Beliebige Fenstertypen	Kompositionsänderungen	Algorithmenänderungen	Parameteränderungen	Verteilt anwendbar	Migrationsdauer
ideal	○	○	-	-	×	×	×	×	×	×	×	optimal
MS	-	×	×	×	×	-	-	×	-	-	-	kurz
GMS	-	×	×	×	×	×	-	×	-	-	(×)	kurz
PT	×	-	○	×	-	-	-	×	-	-	-	lang
GPT	×	-	○	-	×	×	-	×	-	-	×	lang
GenMig	×	-	○	-	×	-	-	×	-	-	-	lang
HybMig	×	×	×	-	×	-	-	×	-	-	-	lang
HybMig mit BSC	×	×	-	-	×	-	-	×	-	-	-	kurz
GHM	×	×	×	-	×	×	-	×	-	-	(×)	kurz
OSIRIS-SE	-	∗	×	×		×	×	-	×	-	×	
StreamMine	-	∗	×	×		×	×	-	-	-	×	
Aurora	-	-	○	×		×	×	‡	-	-	-	
Medusa	-	×	×	×		×	×	‡	-	-	×	
Borealis	×	∗	×	×		×	×	‡	-	×	×	
STREAM (StreaMon)	-	-	○	×		-		×	-	×	-	
TelegraphCQ (Flux)	×	×	×	×		×	×	-	-	-	×	
NexusDS						×		×	×	×	×	
ACDS		×						-	-	×	×	

Tabelle 3.2: Eigenschaften existierender Migrationsstrategien (erste Gruppe) und anderer adaptiver Datenstromsysteme (zweite Gruppe)
Legende: × – ja, - – nein, ○ – nicht relevant, (×) – mit Anpassungen, ∗ – aus Sicherungspunkten, ‡ – nur zwischen Verbindungspunkten

Migration ein Zustandstransfer verwendet wird, werden lediglich notwendige Tupel aus den Zuständen übertragen.

Existierende Migrationsstrategien Die Analyse der existierenden Migrationsstrategien zeigt, dass alle Strategien Kompositionsänderungen unterstützen, Algorithmen- und Parameteränderungen jedoch nicht. Außerdem sind alle außer GPT in der beschriebenen Form nur in zentralisierten Datenstromsystemen anwendbar. Ursache dafür ist meistens die Verwendung von gemeinsamen Warteschlangen oder gemeinsamen Zuständen. In verteilten Systemen ist dies nicht möglich, und Daten müssen über das Netzwerk übertragen werden, um auf anderen Bearbeitungsknoten verfügbar zu sein. Dieser Transfer benötigt Zeit, welche in den Migrationsstrategien berücksichtigt werden muss. GMS kann mit speziellen Anpassungen, z. B. mit sortierten Warteschlangen, für verteilte Systeme kompatibel gemacht werden, da die Transferzeit zwar die Migrationszeit verlängert, sich aber nicht auf den Ablauf der Strategie auswirkt. Das Gleiche gilt für GHM.

Die Migrationsstrategien MS, GMS und PT sorgen für eine Ausgabeverzögerung, was ein wesentlicher Nachteil ist. Alle anderen werden zu Migrationsbeginn kurz pausiert, um die Anfragen für die Migration vorzubereiten, z. B. Eingangsströme mit der neuen Anfrage verbinden oder zusätzliche Operatoren einfügen. Ebenso kann am Migrationsende eine kurze Unterbrechung nötig sein, um zusätzlich Operatoren wieder zu entfernen und den Endzustand der Anfrage herzustellen. Diese Unterbrechungen der Verarbeitung, die ebenso eine Ausgabeverzögerung hervorrufen können, sind im Vergleich zu denen bei MS, GMS und PT aber marginal, weshalb sie in der Tabelle nicht berücksichtigt wurden.

Weiterhin ist zu beobachten, dass bei der Verwendung von Zustandstransfer, mit Ausnahme von HybMig mit BSC, der Zustand immer vollständig übertragen wird, d. h. gegebenenfalls auch Tupel, die nie zu einem Ergebnis beitragen. Die Strategien GMS, GPT und GHM sind in der Lage, Anfragen mit beliebigen Operatoren zu migrieren. Der Grund dafür ist die Betrachtung der Anfragen als Black Box. Für beliebige Fenstertypen ist keine der Migrationsstrategien geeignet. Meistens ist die Beschreibung auf zeitbasierte, gleitende Fenster beschränkt. Für alle Migrationsstrategien ist die Migrationsdauer qualitativ angegeben. Insbesondere PT, GPT, GenMig und HybMig weisen eine lange Migrationsdauer auf, welche exakt der Anlaufzeit der Anfrage entspricht.

Andere adaptive Datenstromsysteme Die Gruppe der anderen Systeme umfasst vor allem verteilte Systeme, die Funktionen zur Ausfallsicherheit, Wiederherstellung und Optimierung von Anfragen im laufenden Betrieb besitzen. Anfragemodifikationen wurden von OSIRIS-SE, Aurora, Medusa, Borealis, StreaMon, NexusDS und ACDS gezeigt. Beim System Aurora und dessen Nachfolgern ist die Modifikation von Verbindungspunkten abhängig und somit beschränkt. Im Unterschied dazu, können bei StreaMon Verbundoperatoren beliebig umsortiert werden. Bei OSIRIS-SE werden ausschließlich Algorithmenwechsel unterstützt. In Borealis, StreaMon und ACDS ist es möglich, Parameter zu ändern. In Borealis können damit auch einfache Algorithmen modifiziert wer-

den. Beliebige Aktualisierungen, wie Änderungen des Quellcodes, sind damit aber nicht realisierbar. Lediglich NexusDS bietet die volle Flexibilität, Parameter, Algorithmen und Komposition von Anfragen zur Laufzeit zu verändern. Systeme, die keine Anfragemodifikationen vorsehen, sind StreamMine und TelegraphCQ mit Flux. Der Schwerpunkt liegt dort auf der Parallelisierung von Anfragen.

Die Wiederherstellung wird bei den meisten Systemen aus zuvor abgespeicherten Sicherungspunkten durchgeführt, d. h. einzelne Operatoren werden mit einem vorherigen Zustand initialisiert. Deshalb sind die Methoden für beliebige Operatoren und Fenstertypen anwendbar. Sofern der Zustand gesichert wurde, wird er auch vollständig übertragen und wieder hergestellt. Ausnahmen bilden dabei ACDS und Flux, beide transferieren Zustände aus laufenden Operatoren. Für alle Systeme gilt, dass für die Dauer der Wiederherstellung oder der Optimierung keine Eingangswerte von den betroffenen Operatoren und Anfragen verarbeitet werden, sodass es zu einer Verzögerung der Verarbeitung kommt, und somit Ergebnisse verspätet ausgegeben werden. Ausgenommen von dieser Betrachtung ist NexusDS, da keine Details zur Migrationsdurchführung beschrieben wurden. Aus diesem Grund kann auch keine Aussage zu den Konsequenzen der Migration getroffen werden.

Eine Migrationsdauer ist für die Systeme der zweiten Gruppe nicht angegeben, da es sich nicht um Migrationsstrategien handelt. Auch eine Bewertung der Eigenschaft „korrekte Reihenfolge“, die nur auf Migrationsstrategien zutrifft, wurde für diese Gruppe ausgelassen. Trotzdem implementieren die Datenstromsysteme dieser Gruppe interessante Konzepte, die auch für Migrationsaufgaben angewandt werden können.

3.5 Zusammenfassung

Existierende Migrationsmethoden für Datenstromanfragen verwenden in den häufigsten Fällen eines der beiden Grundprinzipien, Parallelmethode oder Zustandstransfer. Daraus resultieren die Nachteile der Strategien. Entweder die Migration dauert lange, da für die neue Anfrage eine Anlaufzeit berücksichtigt werden muss, in der die inneren Zustände vollständig gefüllt werden, oder die Ausführung der Anfrage wird für die Dauer der Migration pausiert und es werden keine Ergebnisse ausgegeben.

Zur Vermeidung dieser Nachteile wurde eine Strategie entwickelt, die beide Prinzipien kombiniert. Als Ergebnis kann eine bestehende Anfrage in eine optimierte Anfrage überführt werden, bei gleichzeitiger Zustandserhaltung und garantierter Ergebnisausgabe. Diese Lösung beinhaltet aber ein Problem, welches auch für die anderen Ansätze gilt. Die Strategie kann in der beschriebenen Form nur für zentralisierte Systeme verwendet werden, da z. B. gemeinsame Zustände oder gemeinsame Warteschlangen genutzt werden. In verteilten Umgebungen ist ein schneller Zugriff darauf nicht möglich. Da der Transfer im Netzwerk langsamer abläuft als im Arbeitsspeicher, muss eine entsprechende Transferzeit berücksichtigt werden.

Zwar wurden verallgemeinerte Lösungen für jede Strategie entwickelt, die sich prinzi-

piell in einer verteilten Umgebung anwenden lassen. Jedoch wird dabei der vollständige Zustand übertragen, was dazu führt, dass unter Umständen Zustandswerte übertragen werden, die nie zu einem Ergebnis beitragen.

Die Kombination der Grundprinzipien ist der richtige Ansatz für eine schnelle Anfragenmigration bei gleichzeitiger, garantierter Ergebnisausgabe. Was fehlt, ist die Beschränkung des Zustandstransfers auf notwendige Zustandswerte und eine Berücksichtigung der Zustandstransferzeit im Netzwerk, um eine Anwendung in verteilten Umgebungen und eine Optimierung der Migrationsdauer zu ermöglichen.

Kapitel 4

Adaptionsmethodik für laufende Datenstromanfragen

Dieses Kapitel beschreibt eine allgemeine Adaptionsmethodik für laufende Datenstromanfragen bestehend aus drei Schritten, welche nachfolgend anhand eines durchgehenden Beispiels erklärt werden. Des Weiteren wird der Zweck des Zustandstransfers betrachtet und die Grundidee seiner Durchführung erläutert.

Veränderliche Bedingungen erfordern eine anpassungsfähige Verarbeitungslogik. Ausgehend von einer Originalanfrage und deren Konfiguration ist das Ziel eine laufende Zielanfrage, d. h. eine modifizierte oder neue Anfrage. Ziel dieser Arbeit ist die Unterstützung des Übergangs von Originalanfrage zur Zielanfrage. Die Anfrageoptimierung und das Finden der Zielanfrage stehen nicht Mittelpunkt.

Prinzipiell können alle Änderungen direkt an der Originalanfrage durchgeführt werden. Allerdings kann dies zu unerwünschten Effekten führen, beispielsweise zu einer Verzögerung von Ausgabewerten. Vor allem die Existenz von zustandsbehafteten Operatoren in der zu verändernden Anfrage beeinflusst die Migration, da gegebenenfalls die Anlaufzeiten dieser Operatoren berücksichtigt werden müssen. Beim Übergang von der alten zur neuen Anfrage kann es so zu undefinierten Situationen kommen.

Zur Vermeidung dieser unerwünschten Effekte wird eine Adaptionsmethodik benötigt, welche die Anpassung an einer Anfrage mit einem möglichst geringen Einfluss auf die zu ändernde Anfrage selbst durchführt. Für alle unbeteiligten Anfragen und Systemelemente darf die Durchführung einer Änderung nicht wahrnehmbar sein. Zu diesem Zweck wird das Gesamtsystem logisch in Primärsystem und Testsystem unterteilt. Alle operativen Anfragen sind dem Primärsystem zugeordnet. Im Testsystem können neue, experimentelle Systemelemente und Anfragen angelegt und untersucht werden. Ausgabewerte des Testsystems dürfen aber keinesfalls in das Primärsystem gelangen. In umgekehrter Richtung ist die Kommunikation erlaubt, d. h. es dürfen Daten aus dem Primärsystem ins Testsystem übertragen werden, beispielsweise von Datenquellen. Alle Aktivitäten im Testsystem werden gegenüber dem Primärsystem mit einer geringeren Priorität ausgeführt, um die Verarbeitung im Primärsystem nicht zu beeinflussen.

Die physische Verteilung von Operatoren ist vor allem in verteilten, heterogenen Umgebungen zu betrachten. Bei der Optimierung einer Anfrage ist dann zu untersuchen,

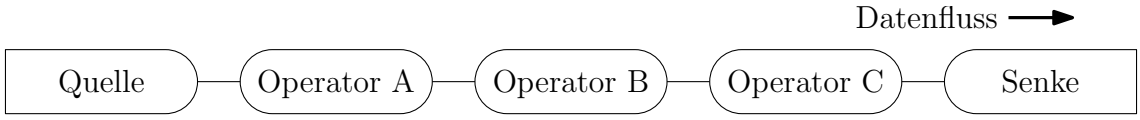


Abbildung 4.1: Originalanfrage vor der Modifikation

ob die verfügbaren Ressourcen, z. B. CPU und Arbeitsspeicher, auf den entsprechenden Bearbeitungsknoten für die Ausführung der neuen Operatoren ausreichend sind. Existierende Verfahren zur Verteilungsplanung in heterogenen Umgebungen, z. B. [AWHK07], können dabei behilflich sein. Eine Kombination von Verteilungsplanungsmethoden mit Migrationsverfahren erscheint interessant, ist jedoch nicht Gegenstand dieser Arbeit. Stattdessen wird in jedem Fall eine hinreichende Verfügbarkeit von Ressourcen angenommen.

Die Adaptionsmethodik lässt sich in drei Abschnitte unterteilen – Duplikation, Modifikation und Reintegration. Die einzelnen Abschnitte sind nachfolgend im Detail beschrieben. Als Beispiel dient eine einfache, sequenzielle Anfrage wie in Abbildung 4.1. Die Adaptionsmethodik ist allerdings nicht auf sequentielle Anfragen beschränkt. Die Beispielanfrage besteht aus drei Operatoren, die Datenstromwerte einer Datenquelle nacheinander verarbeiten und schließlich an eine Datensenke übermitteln.

Gemäß der ersten Anforderung (Abschnitt 2.5) wird ein aktives Verarbeitungsmodell, insbesondere eine aktive, Push-basierte Kommunikation der Systemelemente, durch Publish-Subscribe realisiert. Publish-Subscribe entspricht dem Observer-Entwurfsmuster [GoF94] und ermöglicht eine Entkopplung der Kommunikation der Systemelemente. Die Systemelemente können dabei die Rolle eines Senders[1] (Datenquellen) oder eines Empfängers[2] (Datensenken) einnehmen. Operatoren implementieren beide Rollen. Ein Empfänger kann sein Interesse am Ausgangsdatenstrom eines Senders bekanntgeben, indem er sich bei diesem anmeldet. Der Sender benachrichtigt alle registrierten Empfänger, sobald neue Daten, d. h. neue Verarbeitungsergebnisse, vorliegen.

Eine weitere Entkopplung ist realisierbar, wenn sich ein Empfänger nicht direkt beim Sender anmeldet, sondern bei einer zentralen Registrierung. Dafür wird im System jeder Datenstrom mit einem eindeutigen Bezeichner, einer Datenstrom-Identifikationsnummer (DSID), versehen. Ein Empfänger teilt das Interesse an einer bestimmten DSID der Registrierung mit. Ein Sender, Produzent eines Ausgangsdatenstroms mit einer bestimmten DSID, kann alle Interessenten seiner DSID von der Registrierung beziehen, um diese bei neuen Daten zu benachrichtigen. Der Vorteil dieser Erweiterung ist, dass sich Empfänger nicht direkt mit den Sendern in Verbindung setzen müssen. Aus Sicht der Empfänger können dadurch Sender transparent ersetzt werden, die Erhaltung der DSID vorausgesetzt.

[1] eigentlich Subjekt (engl. „Publisher“)
[2] eigentlich Beobachter (engl. „Observer“)

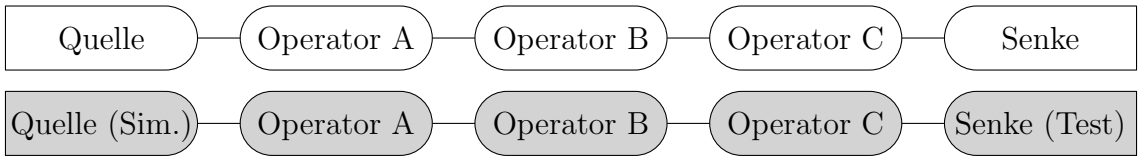


Abbildung 4.2: Duplikation der Originalanfrage in eine Testumgebung (graue Systemelemente)

4.1 Duplikation

Die erste Phase der Anpassung ist das Erstellen einer neuen Anfrage. Eine einfache Möglichkeit ist das Duplizieren der Originalanfrage. Dabei wird im Testsystem eine Anfrage mit der gleichen Konfiguration der Originalanfrage erstellt. Ein Beispiel findet sich in Abbildung 4.2. Zur Originalanfrage gibt es ein Duplikat im Testsystem, gekennzeichnet durch graue Systemelemente.

Bei der Duplikation ist zu beachten, dass die Ausgänge aller Operatoren der duplizierten Anfrage keine Daten ins Primärsystem ausgeben. Die neuen Operatoren werden also untereinander im Testsystem verbunden. Statt der Datensenke des Primärsystems kann eine Datensenke im Testsystem angelegt werden. Diese Testdatensenke kann in ihrer Funktionalität von der originalen Senke abweichen. So ist es möglich, eingehende Daten zusätzlich zu analysieren oder an geeignete Systeme weiterzusenden, z. B. an ein Beobachtungssystem oder eine Visualisierung. Die Datenquelle der duplizierten Anfrage kann entweder die Quelle der Originalanfrage sein oder eine Quelle im Testsystem, beispielsweise eine Quelle, die mit einem gleichartigen Sensor verbunden ist. Es ist auch möglich, eine Quelle zu simulieren (Abbildung 4.2). Simulierte Quellen enthalten entweder Simulationsalgorithmen, um Eingangsdaten im DSMS zu erzeugen, oder sind mit externen Simulationssystemen verbunden. Die Verwendung einer simulierten Quelle bietet unter Umständen den Vorteil, dass Eingangswerte mit einer höheren Frequenz erzeugt und somit schneller getestet werden können. Dies kann das Finden einer optimierten Anfrage beschleunigen. Neben simulierten Daten können zum Testen auch historische Daten hilfreich sein, sofern sie vorher aufgezeichnet wurden. Eine Quelle kann diese Daten dann laden und als Datenstrom wiedergeben.

Neben der Duplikation besteht die Möglichkeit, Anfragen im Testsystem neu zu erstellen oder gespeicherte Anfragenkonfigurationen zu laden. Ebenso können einzelne Operatoren oder Teilanfragen erstellt oder dupliziert werden. Bei Bedarf ist eine Mehrfachduplikation möglich. Dies kann dazu dienen, mehrere Alternativen zu entwickeln und zu vergleichen.

4.2 Modifikation

Alle Elemente des Testsystems können direkt modifiziert und getestet werden. Änderungen können durch Anwender oder automatisch durch Optimierungsmethoden durchgeführt werden, wenn entsprechende Schnittstellen existieren. Die verschiedenen Möglichkeiten, eine Anfrage zu modifizieren, sind Parameteränderungen, Algorithmusänderungen und Änderungen der Anfragenkomposition. Modifikationen können durch die Verwendung geeigneter Datensenken oder durch angeschlossene Systeme analysiert werden. Außerdem ist es möglich, zusätzliche Informationen zu protokollieren und für eine Auswertung zur Verfügung zu stellen. Duplikation und Modifikation können gemischt ausgeführt werden, um iterativ Anfragen zu erstellen und zu verbessern. Wie bereits erwähnt, wird der Prozess der Anfrageoptimierung selbst nicht betrachtet. Vielmehr wird von der Existenz einer optimierten Anfrage ausgegangen und die Integration in ein bestehendes System beschrieben. Es ist durchaus möglich, dass sich durch Änderungen auch die Semantik einer Anfrage ändert, d. h. alte und neue Anfrage sind semantisch nicht äquivalent.

4.2.1 Modifikation von Parametern

Zur Anfragenanpassung können Parameteränderungen benutzt werden, indem Systemelemente oder Algorithmen an vorgesehenen Stellen modifiziert werden. Das erfordert, dass Systemelemente oder Algorithmen parametrierbar implementiert sind. Zudem sind Schnittstellen notwendig, um die Parametrierung durchzuführen. Da sich die Funktionalität nur im Rahmen der vorgesehenen Parameter ändern lässt, sind die Änderungsmöglichkeiten beschränkt. Das System Borealis unterstützt Parameteränderungen von Datenstromoperatoren zur Laufzeit, es besitzt jedoch keine Migrationsstrategien. Von existierenden Migrationsverfahren wurden Parameteränderungen nicht beschrieben.

Beispiele für Parameteränderungen sind das Anpassen eines Grenzwertes eines Filteroperators, das Vergrößern oder Verkleinern der Fenstergröße, oder das Variieren der Abtastfrequenz einer Datenquelle zur Anpassung des Durchsatzes an die aktuelle Systemlast. Am Beispiel der Veränderung der Fenstergröße in Abschnitt 2.4 wurde bereits gezeigt, dass sich Anfragenänderungen auf den gesamten Migrationsprozess auswirken und unter Umständen auch Änderungen an nachfolgenden Operatoren erfordern. In Abbildung 4.3 wurden die Operatoren A und B der Testanfrage zu A* und B* modifiziert. Dabei könnte B* eine solche bedingte Änderung darstellen.

Zum Testen kann ein weiteres Duplikat genutzt werden, welches die unveränderte Originalanfrage im Testsystem repräsentiert (Abbildung 4.4). Dank Publish-Subscribe können Datentupel an beliebig viele Nachfolger gesendet werden, so auch von der Simulationsquelle an die Operatoren A und A* im Testsystem. Dies ermöglicht einen direkten Vergleich im Testsystem zwischen der aktuellen Logik und der modifizierten Logik. Insbesondere bei der Verwendung einer simulierten Quelle, die einen höheren Durchsatz als die Originalquelle besitzt, lassen sich schnell Ergebnisse von verschiedenen Anfragen

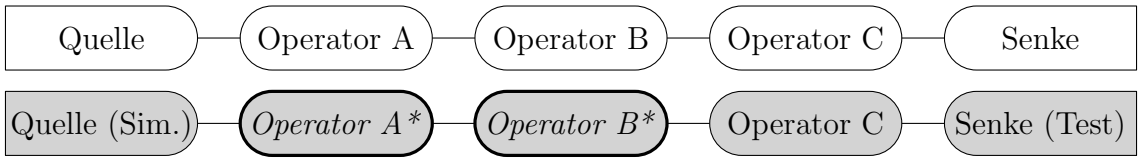


Abbildung 4.3: Modifikation der duplizierten Anfrage

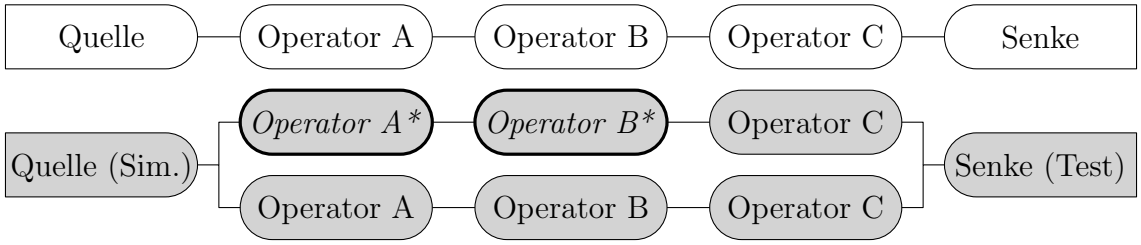


Abbildung 4.4: Mehrfachduplikation zum Vergleich einer modifizierten Anfrage mit einem Duplikat der Originalanfrage im Testsystem

bewerten und Alternativen miteinander vergleichen, vorausgesetzt, dass ausreichend Systemressourcen vorhanden sind.

4.2.2 Modifikation von Algorithmen

Abgesehen von Parameteränderungen, können Algorithmen auch durch Änderungen am Code modifiziert werden oder durch effizientere Algorithmen ersetzt werden. Insofern lässt sich im Vergleich zur Parametrierung ein Operator vollständig und ohne Einschränkungen ändern. Ein Algorithmenaustausch kann über geeignete Schnittstellen und einen Anpassungsmechanismus realisiert werden. Um beliebige Algorithmen für einen Austausch zur Verfügung zu haben, muss eine Möglichkeit bestehen, ausführbaren Programmcode von neuen Algorithmen zur Laufzeit zu laden. Der Austausch von Algorithmen wurde ebenfalls bei OSIRIS-SE als Anpassungsfunktionalität genannt, ist jedoch auf existierende Algorithmen beschränkt. In Borealis sind parametrierbare Algorithmen in einer Datenbank gespeichert. Ob sich die Datenbank zur Laufzeit aktualisieren lässt, ist nicht beschrieben. Zudem sind diese Algorithmen auf Parametrierung begrenzt.

Im Unterschied zu existierenden Systemen werden Algorithmen in einem aktualisierbaren Archiv aufbewahrt. Dieses Archiv kann zur Laufzeit mit neuen Algorithmen ergänzt werden oder existierende Algorithmen können aktualisiert werden. Da Operatoren Instanzen von Algorithmen referenzieren, kann die Aktualisierung des Algorithmenarchivs ohne Einfluss auf die Verarbeitung durchgeführt werden. Der Austausch eines Algorithmus im Operator erfolgt dann durch das Ändern der Referenzierung auf eine neue Algorithmusinstanz. Alte Versionen von Algorithmen bleiben solange als Instanz erhalten, bis sie explizit ausgetauscht werden. Für die Unterscheidung von Algorithmen wird

eine Versionierung vorgenommen [Beh10].

Obwohl Algorithmen auch im Primärsystem während der Laufzeit ausgetauscht werden können, ist ein Austausch in der Testumgebung meistens weniger riskant und unter Umständen einfacher umsetzbar, da die sichere Ausführung der zu ändernden Testanfrage nicht garantiert sein muss und diese nach einem Algorithmenaustausch neu gestartet werden kann. Für die Verwendung in der Originalanfrage sind dann eine Reintegration und gegebenenfalls ein Zustandstransfer durchzuführen. Die Abbildungen 4.3 und 4.4 gelten für Algorithmen- und Parameteränderungen gleichermaßen.

Folgendes Beispiel soll den Ablauf einer Algorithmenanpassung verdeutlichen. Ein Aggregationsoperator nutzt für die Berechnung einer gleitenden Summe ein Fenster von Werten und referenziert einen Algorithmus, der für jedes Ergebnis alle Werte des Fensters summiert. Eine optimierte Version des Algorithmus bildet eine fortlaufende Summe, indem verworfene Werte von der letzten Summe subtrahiert werden und der Eingangswert addiert wird. Für die Realisierung ist ebenfalls ein modifizierter Operator notwendig. Da Algorithmen zustandsfrei sind und die notwendigen Daten durch Parameterübergabe erhalten, muss der Operator zusätzlich zum Fenster das letzte Ergebnis in seinem inneren Zustand speichern. Beim Eintreffen eines neuen Eingangstupels wird dann nicht mehr das komplette Fenster an den Algorithmus übergeben, sondern während der Aktualisierung des Fensters werden die verworfenen Werte gespeichert und nach der Aktualisierung mit der letzten Summe und dem Eingangstupel an den neuen Algorithmus übergeben. Beide Algorithmusversionen unterscheiden sich also auch in Form und Anzahl der übergebenen Werte. Trotz der Anpassung des Operators liegt keine Kompositionsänderung vor, da der Operator weiterhin mit dem gleichen Vorgänger und den gleichen Nachfolgern verbunden ist.

Ein weiteres Beispiel ist der zeitweilige Austausch von Algorithmen in Operatoren. Diese Funktionalität kann genutzt werden, um das System in Phasen von erhöhter Last durch den Einsatz von schnelleren aber weniger genauen Algorithmen zu entlasten. Dafür wird der Operator der Originalanfrage direkt modifiziert. Da bei diesem Anwendungsfall approximierte Ergebnisse erzeugt werden, wird er nicht weiter diskutiert. Trotzdem ist eine Realisierung mit der beschriebenen Methodik möglich. Ein Algorithmenaustausch findet dann zweimal statt – zu Beginn und am Ende der Lastphase.

4.2.3 Modifikation der Anfragenkomposition

Der letzte Modifikationstyp ist die Veränderung der Anfragenkomposition. Es besteht die Möglichkeit, die Reihenfolge von Operatoren innerhalb der Anfrage zu vertauschen. Ebenso können Operatoren oder Teilanfragen eingefügt, entfernt oder ersetzt werden, beispielsweise der Austausch von mehreren Operatoren durch einen einzelnen, effizienteren Operator. Das Beispiel in Abbildung 4.5 zeigt beide Fälle. Im Testsystem existiert eine Anfrageversion mit Operator B vor Operator A und eine weitere Version mit Operator D als Ersatz für die Operatoren A und B. Beide Anfrageversionen sind für eine Evaluierung mit derselben Datensenke verbunden und beziehen ihre Eingangsdaten von

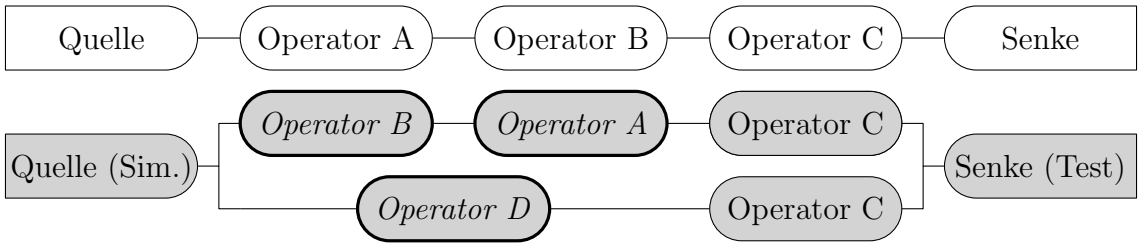


Abbildung 4.5: Zwei alternative Kompositionsänderungen im Testsystem

der Simulationsdatenquelle.

Das Umsortieren von Operatoren ist ein häufig genutztes Mittel, um Anfragen zu optimieren, und wird von allen bekannten Migrationsstrategien unterstützt. In Abschnitt 3.1 wurde ein Beispiel für Bäume von Verbundoperatoren gezeigt. Ein weiteres Beispiel ist das Verschieben von Filter-Operatoren in Richtung Datenquelle. Durch die Platzierung am Anfang der Anfrage können irrelevante Werte frühzeitig gefiltert werden. Die Anfrage kann dadurch mit einer geringeren Last ausgeführt werden. Für einen Zustandstransfer ist eine solche Verschiebung zu berücksichtigen.

Das Einfügen von Operatoren ist weniger gebräuchlich, ist aber notwendig, wenn Anfragen zur Laufzeit an neue Gegebenheiten angepasst werden sollen, z. B. das Einbeziehen zusätzlicher Datenquellen. Stellt man sich einen Anwendungsfall vor, in dem ein Sensor zur Erhöhung der Zuverlässigkeit mit weiteren Sensoren ergänzt wird, um einen fehlertoleranten 2oo3-Sensor[3] zu realisieren, dann muss auch die Bearbeitungslogik an diese Situation angepasst werden. Im DSMS nehmen neue Datenstromquellen die Verbindung zu den neuen Sensoren auf, und ein neuer, zusätzlicher Operator übernimmt die 2oo3-Logik. Der Ausgangsdatenstrom des neuen Operators ersetzt den Ausgangsdatenstrom der ursprünglichen Datenquelle. Alle nachfolgenden Operatoren können von der Originalanfrage unverändert übernommen werden. Für sie verläuft der Wechsel transparent.

Im vorherigen Abschnitt wurde der zeitweilige Austausch von Algorithmen beschrieben, indem ein Algorithmus direkt im Originaloperator verändert wird. Eine weitere Lösung für dieses Szenario ist die Verwendung von zwei Operatoren, die je einen der Algorithmen referenzieren. Der Vorteil dieser Realisierung ist, dass beide Operatoren ihren eigenen Zustand haben können und diesen, auch wenn sie nicht Teil der aktiven Anfrage sind, kontinuierlich aktualisieren können. Der Wechsel zwischen den Algorithmen findet dann durch den Austausch der Operatoren statt und ist somit eine Kompositionsänderung, da die Publish-Subscribe-Verbindungen zwischen den Operatoren verändert werden.

[3] 2oo3 (2-out-of-3) ist ein Prinzip zum Erreichen einer Fehlertoleranz. Basierend auf einer Mehrheitswahl werden Signale von drei Eingängen verglichen und zu einem Signal vereint [Nel90].

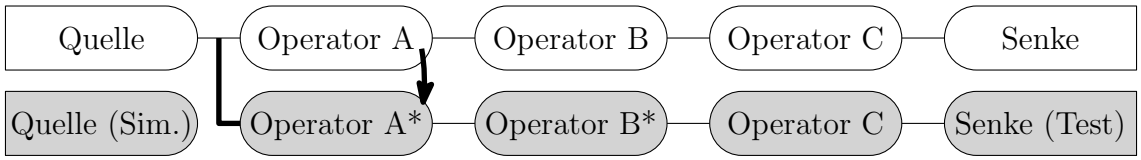


Abbildung 4.6: Reintegrationsbeginn und Zustandstransfer

4.3 Reintegration

Nach dem Testen und Bewerten von alternativen Anfragen im Testsystem sind die notwendigen Änderungen an der Originalanfrage durchzuführen. Eine Änderung muss nicht alle Elemente einer Anfrage beeinflussen. Bezieht sich eine Änderung auf Eigenschaften einzelner Elemente, bleiben die meisten Elemente einer Anfrage unverändert. Während der Reintegration werden also die Anfrage, Teile der Anfrage oder einzelne Operatoren ausgetauscht, d. h. die Migration von der alten Anfrage zur neuen Anfrage wird durchgeführt.

Da die neue Anfrage bereits im Testsystem existiert, kann diese direkt benutzt werden. In diesem Fall ist sie von den Datenquellen im Testsystem zu trennen und alle inneren Zustände sind zu leeren, damit keine Vermischung mit den Daten des Primärsystems auftritt. Soll die Anfrage im Testsystem erhalten werden, kann durch Duplikation die gleiche Anfrage nochmals erstellt werden. Dieser Ablauf gilt für Teilanfragen und einzelne Operatoren analog. Als nächstes wird die Verbindung zur Datenquelle des Primärsystems oder dem Ausgang des entsprechenden Vorgängers hergestellt. Die erste Konstellation ist in Abbildung 4.6 dargestellt.

Für zustandsbehaftete Operatoren beginnt dann der Zustandstransfer. Für den Fall, dass die Anfrage im Testsystem bereits mit der Datenquelle des Primärsystems verbunden war, kann auf ein Verwerfen der Zustände zu Beginn der Reintegration verzichtet werden. Der Zustandstransfer entfällt gegebenenfalls oder wird verkürzt, ebenso die Migration.

Für den Umschaltzeitpunkt können verschiedene Bedingungen festgelegt werden. Für Anwendungsfälle wie Bäume von Verbundoperatoren ist eine sinnvolle Bedingung, wenn die inneren Zustände der neuen Anfrage vollständig gefüllt sind, d. h. Umschalten bei Migrationsende. Die neue Anfrage ist dann in der Lage, semantisch äquivalente Ergebnisse zur Originalanfrage zu produzieren. Bei der Vergrößerung von Fenstern ist es möglich, zur neuen Anfrage zu wechseln, sobald das neue Fenster größer ist als das des Originaloperators, da zu erwarten ist, dass dann genauere Ergebnisse produziert werden können als mit dem Originaloperator. Für andere Fälle ist es möglich, zur neuen Anfrage zu wechseln, sobald diese alle notwendigen Zustandswerte erhalten hat, um korrekte Ergebnisse zu produzieren. Der Zustandstransfer wird bis zum Migrationsende fortgeführt, um die inneren Zustände zu vervollständigen. Weitere Bedingungen sind denkbar und können durch Benutzer spezifiziert werden.

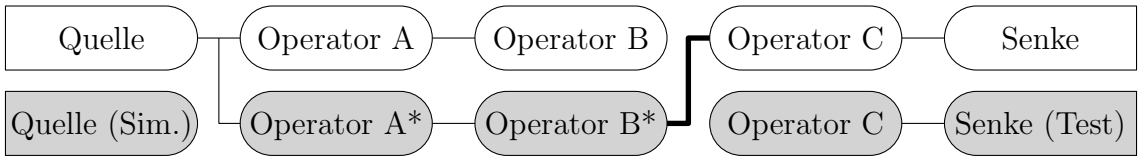


Abbildung 4.7: Reintegration modifizierter Elemente

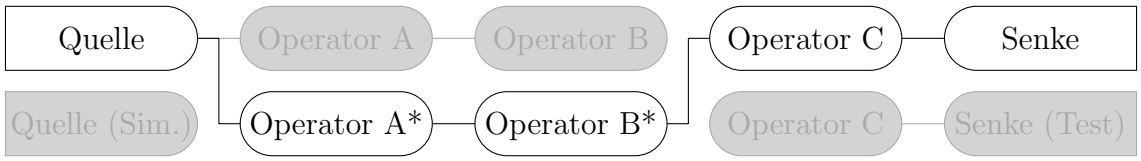


Abbildung 4.8: Modifizierte Anfrage nach der Migration

Sobald die Umschaltbedingung erfüllt ist, muss zwischen den entsprechenden Operatoren der Umschaltvorgang erfolgen. Dabei ist darauf zu achten, dass keine doppelten Ergebnisse übermittelt werden und dass keine Ergebnisse fehlen. Auch die Reihenfolge der Ergebnisse ist einzuhalten und eine Verzögerung ist zu vermeiden. Die Umschaltung erfolgt, sofern möglich, transparent für unveränderte Nachfolgeoperatoren. In Abbildung 4.7 ist der Abschluss der Reintegration für die Beispielanfrage veranschaulicht. Die Ausgänge von Operator B und Operator B* werden umgeschaltet. Operator C nimmt den Wechsel nicht wahr. Durch die Verwendung von Publish-Subscribe sind die Operatoren entkoppelt. Operator C hat die Ergebnisse einer bestimmten DSID abonniert, die mit der Umschaltung von Operator B* gesendet werden und nicht mehr von Operator B. Mit der Umschaltung kann eine automatische Versionierung von Operatoren erfolgen [Beh10].

Der Zustand der Anfrage nach der Migration ist in Abbildung 4.8 dargestellt. Die Anfrage enthält die modifizierten Operatoren. Die Simulationsquelle, Operator C und die Datensenke im Testsystem können verworfen werden. Die alten Operatoren A und B können ebenfalls entfernt werden oder bei Bedarf erhalten werden. Sie können genutzt werden, um zu einem späteren Zeitpunkt zurück zur alten Anfrage zu wechseln. In diesem Beispiel ist Operator A weiterhin mit der Datenquelle verbunden. Die Verarbeitung des Operators kann deaktiviert werden, so dass lediglich sein Zustand durch neue Datenstromwerte aktualisiert wird. Sofern sich hinter Operator A kein zustandsbehafteter Operator befindet, ist ein unmittelbarer Wechsel durch Reintegration dieser Teilanfrage möglich. Anderenfalls sind auch die Zustände der nachfolgenden Operatoren zu aktualisieren, indem die Verarbeitung aller Operatoren rechtzeitig aktiviert wird oder, falls möglich, durch einen Zustandstransfer.

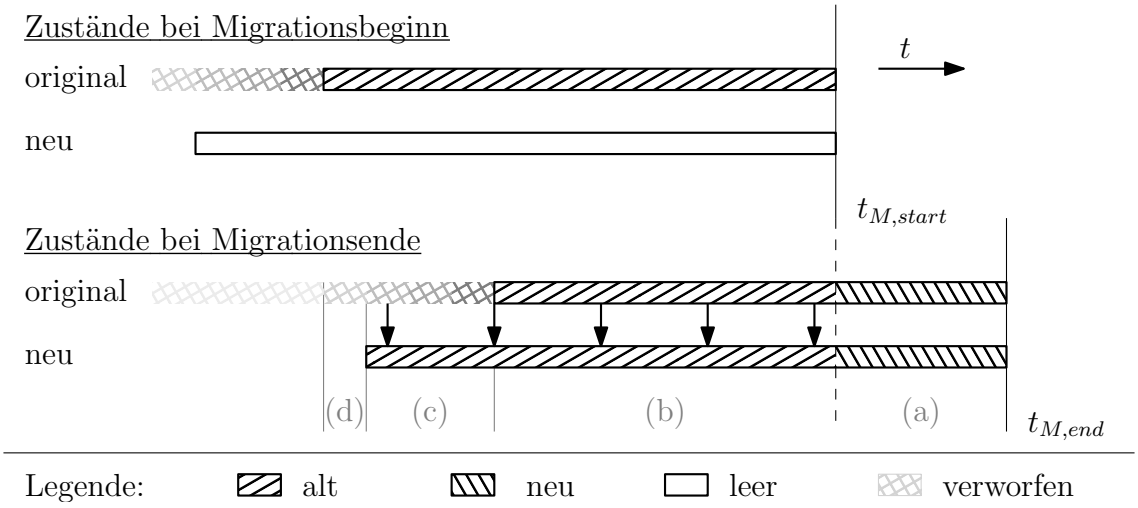


Abbildung 4.9: Grundidee des Zustandstransfers

4.4 Zustandstransfer

Der Zustandstransfer ist ein wichtiges Werkzeug zur Beschleunigung der Migration von zustandsbehafteten Operatoren, ist aber generell unabhängig von der Migration. Ein Zustandstransfer kann während jeder Phase durchgeführt werden. Überträgt man den Zustand mit der Duplikation, ist es möglich, mit initialisierten Anfragen im Testsystem zu beginnen. In der Modifikationsphase kann es nützlich sein, die geänderte Anfrage mit Daten des Primärsystems zu testen. Ein Zustandstransfer hilft, gültige Ergebnisse eher auszugeben. Die Verwendung während der Reintegration wurde bereits gezeigt. Wurde der Zustand bereits in einer der ersten Phasen übertragen und ist auch nach den erfolgten Änderungen gültig, kann direkt zu der geänderten Anfrage gewechselt werden.

Im Verlauf des Zustandstransfers nehmen die beteiligten Operatoren unterschiedliche Rollen ein und haben unterschiedliche Aufgaben zu erfüllen. Der Originaloperator agiert als Zustandssender, der neue Operator ist der Zustandsempfänger. Der Zustandssender hat die Aufgabe, Werte aus seinem inneren Zustand entsprechend einer Zustandstransferstrategie auszuwählen und diese an den Zustandsempfänger zu übertragen. Beide Rollen erfordern funktionale Erweiterungen der entsprechenden Operatoren während der Phase des Zustandstransfers.

Die Durchführung eines Zustandstransfers benötigt Zeit. Dies gilt insbesondere in verteilten Datenstromsystemen, da die Zustandswerte über das Netzwerk übertragen werden. Da währenddessen neue Werte an den Eingängen der Operatoren eintreffen können, müssen zur Vermeidung von Fehlfunktionen Methoden zur Synchronisation zwischen regulärer Verarbeitung und Zustandstransfer genutzt werden. Die Beschränkung des Transfers auf notwendige Werte minimiert die Transferzeit, wovon besonders die Migration in verteilten Szenarien profitiert.

Die Grundidee des Zustandstransfers ist in Abbildung 4.9 dargestellt. Es sind jeweils die Zustände des Originaloperators und des neuen Operators bei Migrationsbeginn ($t_{M,start}$) und Migrationsende ($t_{M,end}$) dargestellt. Die durchgeführte Änderung ist eine Vergrößerung des Fensters. Durch den Transfer von Zustandswerten kann eine schnellere Migration erreicht werden. Bei Migrationsende setzt sich ein Zustand des neuen Operators aus neuen Werten des Eingangsstroms (Anteil (a)) und aus alten Werten[4] (Anteile (b) und (c)), die vom Zustandssender übertragen wurden, zusammen. Darunter sind Werte, die beim Zustandssender bereits verworfen wurden (Anteil (c)). Es ist aber auch zu beobachten, dass nicht alle Werte, die bei Migrationsbeginn im Zustand des Zustandssenders verfügbar sind, für die Inbetriebnahme des neuen Operators relevant sind (Anteil (d)). Die Menge der notwendigen Zustandswerte muss also genau bemessen werden. Dafür ist die Kenntnis der Migrationsdauer notwendig.

Damit Zustände zwischen Operatoren übertragen werden können, müssen zuvor geeignete Paare von Zustandssendern und -empfängern ausgewählt werden. Neben einer manuellen Festlegung besteht die Möglichkeit, geeignete Zustandspaare automatisch zu suchen. In jedem Fall ist die Semantik der Original- und Zielanfrage zu berücksichtigen. Dies beinhaltet auch die Berücksichtigung der durchgeführten Änderungen.

Zustandstransferstrategien sowie Konzepte zur Synchronisation, zur funktionalen Erweiterung, zur Bestimmung von Zustandspaaren und zur Auswahl der notwendigen Zustandstransferwerte werden in den nächsten zwei Kapiteln beschrieben.

4.5 Zusammenfassung

Dieses Kapitel stellt eine allgemeine Methodik zur Anpassung von laufenden Datenstromanfragen vor, welche gleichzeitig einen der Beiträge dieser Arbeit repräsentiert. Die wesentlichen Eigenschaften der Anpassungsmethodik können wie folgt zusammengefasst werden.

Mit der logischen Unterteilung des Gesamtsystems in Primär- und Testsystem kann zwischen operativen und experimentellen Anfragen unterschieden werden. Mechanismen und Regeln im Umgang mit experimentellen Anfragen unterbinden negative Einflüsse auf operative Anfragen. Experimentelle Anfragen laufen dadurch entkoppelt von operativen Anfragen und werden mit einer geringeren Priorität ausgeführt. Direkte Änderungen an operativen Anfragen sind nur erlaubt, wenn es der Anwendungsfall erfordert. Ansonsten ist ein dreistufiger Änderungsmechanismus zu verwenden – Duplikation, Modifikation, Reintegration.

Experimentelle Anfragen werden durch Duplikation existierender Anfragen angelegt. Außerdem ist es möglich, Anfragen neu zu erstellen oder zu laden. Als Datenquellen für experimentelle Anfragen werden die Datenquellen der operativen Anfrage oder adäquate simulierte Datenquellen verwendet. Mittels Mehrfachduplikation können mehrere Anfragen erzeugt und verglichen werden. Modifikationen an den experimentellen Anfragen

[4] Die Verwendung von „neu“ und „alt“ entspricht der Definition von PT (Abschnitt 3.1.2, S. 34)

können in Form von Parameteränderungen, Algorithmusänderungen und Kompositionsänderungen vorgenommen werden. Das Ändern von Algorithmen zur Laufzeit wird durch ein aktualisierbares Algorithmenarchiv unterstützt. Vor allem durch Algorithmus- und Kompositionsänderungen lassen sich Anfragen vollständig und uneingeschränkt modifizieren. Vorhandene, migrationsunterstützende Datenstromsysteme sind diesbezüglich beschränkt. Die Durchführung der Migration findet während der Reintegration statt. Mit Hilfe eines Zustandstransfers kann die Migration beschleunigt werden.

Kapitel 5

Zustandstransfer

In diesem Kapitel wird der Zustandstransfer zunächst für periodische und schwankungsbeschränkte periodische Datenströme betrachtet. Sofern nicht anders angegeben, gelten die vorgestellten Konzepte und Strategien für beide Datenstromtypen. Eine Betrachtung für die Anwendung in aperiodischen Datenströmen folgt im darauffolgenden Kapitel.

Besitzt eine zu migrierende Anfrage mehrere Eingangsdatenströme, so reicht ein periodisches Verhalten aller Eingangsströme nicht aus. Vielmehr ist ein periodisches Systemverhalten notwendig, um die nachfolgenden Berechnungsmodelle anwenden zu können. In Abbildung 5.1 sind Beispiele für periodisches und aperiodisches Systemverhalten qualitativ als Impulsdiagramme dargestellt. Jeder Impuls symbolisiert einen Eingangswert und dessen Auslastung der CPU-Zeit[1], woraus die Periodendauer T und die Verarbeitungszeit T_p resultieren. Die Verarbeitungszeit ist die Zeitspanne zwischen Ankunft des Eingangswertes und Ausgabe des letzten Ergebniswertes für diesen Eingangswert. Es sind jeweils die drei Eingangsdatenströme ($\mathcal{S}_A$, $\mathcal{S}_B$ und $\mathcal{S}_C$) und die resultierende Systemauslastung abgebildet.

Die Varianten (a) und (b) lassen sich z. B. durch zyklisches Abfragen von externen Datenquellen realisieren. Die resultierende Systemauslastung ist in beiden Fällen periodisch. Mit Hilfe der Eigenschaften der einzelnen Eingangsdatenströme können die

[1]Es wird ein einfaches Modell angenommen, in dem die gesamte Kapazität der CPU für die Verarbeitung eines Wertes zur Verfügung steht.

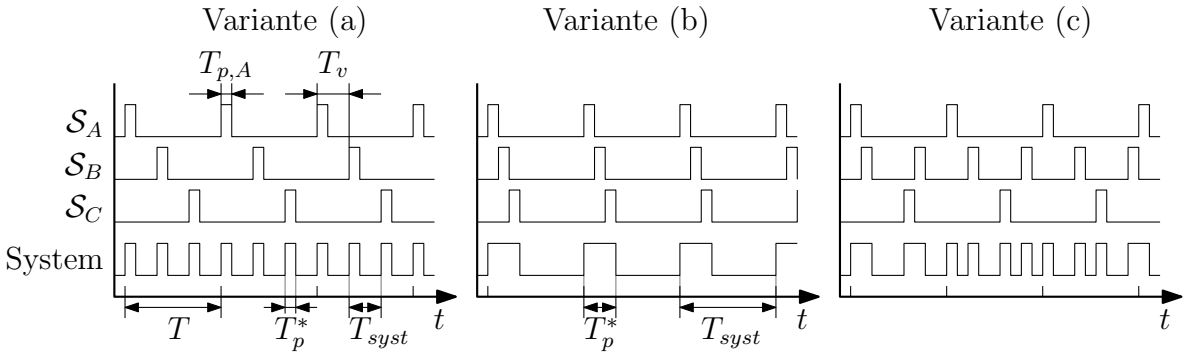


Abbildung 5.1: Periodizität des Gesamtsystems

Systemeigenschaften abgeleitet werden, d. h. Periodendauer des Systems T_{syst} und Verarbeitungsdauer des Systems T_p^*. Für periodische Datenströme lässt sich aus den Systemeigenschaften die Pausendauer des Systems gemäß Gleichung (5.1) berechnen.

$$T_i = T_{syst} - T_p^* \tag{5.1}$$

Im abgebildeten Beispiel enthalten sowohl Variante (a) als auch Variante (b) drei Eingangsdatenströme mit gleicher Periodendauer, d. h. $T = T_A = T_B = T_C$. Bei Variante (a) existiert eine gleichgroße Verschiebung zwischen den einzelnen Eingangsdatenströmen. Die Verschiebung T_v entspricht dem Verhältnis von Periodendauer und der Anzahl der Eingangsdatenströme N_e, d. h. $T_v = T/N_e$ mit $N_e = |\{\mathcal{S}_I\}|$. Für (a) entspricht die Periodendauer des Systems der Verschiebung (Gleichung (5.2a)). In Variante (b) werden die Eingangsdatenströme unmittelbar nacheinander verarbeitet, sodass es keine Verschiebung zwischen ihnen gibt und die resultierende Periodendauer des Systems der Periodendauer der Eingangsdatenströme entspricht (Gleichung (5.2b)). Unter der Annahme einer global konstanten Verarbeitungszeit $T_p = T_{p,A} = T_{p,B} = T_{p,C}$ entspricht die gesamte Verarbeitungszeit pro Periode den Gleichungen (5.3). Der Parameter N_i beschreibt dabei die Anzahl der Werte, die ein Eingangsdatenstrom pro Periode ausgibt. Dieser wird ebenso als global gleich angenommen. In Abbildung 5.1 ist $N_i = 1$. Die Pausendauer für die jeweilige Variante ist unter (5.4) angegeben.

Variante (a)	Variante (b)	
$T_{syst} = T_v = T/N_e$	$T_{syst} = T$	(5.2)
$T_p^* = N_i * T_p$	$T_p^* = N_i * N_e * T_p$	(5.3)
$T_i = T_v - N_i * T_p$	$T_i = T - N_i * N_e * T_p$	(5.4)

Für schwankungsbeschränkte Datenströme verläuft das Ermitteln der Systemeigenschaften prinzipiell ähnlich. Die Gleichungen (5.1) bis (5.4) sind jedoch auf die Eigenschaften der schwankungsbeschränkten Ströme anzupassen. In Abschnitt 2.2.1 wurden diese charakterisiert mit einer theoretischen Periodendauer T, den Schwankungen τ und τ' sowie dem Mindestabstand D (vgl. Abbildung 2.2, S. 11). Die Pausendauer in einem System mit schwankungsbeschränkten Eingangsdatenströmen lässt sich entsprechend Gleichung (5.5) berechnen. Dabei wird von einem systemspezifischen Mindestabstand D_{syst} ausgegangen.

$$T_i = D_{syst} - T_p^* \tag{5.5}$$

Unter der Voraussetzung einer global einheitlichen theoretischen Periodendauer $T = T_A = T_B = T_C$ kann je nach Variante die theoretische Periodendauer des Systems errechnet werden (Gleichungen (5.6)). Dabei wird die theoretische Verschiebungszeit der einzelnen Signale wie bei periodischen Datenströmen berechnet, d. h. $T_v = T/N_e$. Nimmt man des Weiteren auch eine globale Gleichheit der Parameter τ, τ' und D an, so kann

der Mindestabstand des Systems D_{syst} mit Hilfe der Gleichungen (5.7) berechnet werden. Sollte die Verarbeitungszeit auch mit Schwankungen behaftet sein, sodass $T_{p,A} \approx T_{p,B} \approx T_{p,C}$, dann besteht die Möglichkeit für die Berechnung der gesamten Verarbeitungszeit pro Periode (T_p^*, Gleichungen (5.8)) den größten aller Werte zu verwenden, was bedeutet $T_p = max(\{T_{p,A}, T_{p,B}, T_{p,C}\})$. Dieses Prinzip lässt sich ebenso auf die Schwankungen und den Mindestabstand anwenden, falls diese für die einzelnen Datenströme unterschiedlich sein sollten. Es bleibt jedoch bei der Annahme, dass N_i für alle Eingangsdatenströme gleich ist. Schließlich kann mit diesen Parametern die Pausendauer des Systems für Variante (a) nach Gleichung (5.9a) und für Variante (b) nach Gleichung (5.9b) bestimmt werden.

Variante (a)	Variante (b)	
$T_{syst} = T_v = T/N_e$	$T_{syst} = T$	(5.6)
$D_{syst} = T_v - \tau - \tau'$	$D_{syst} = D$	(5.7)
$T_p^* = N_i * T_p$	$T_p^* = N_i * N_e * T_p$	(5.8)
$T_i = T_v - \tau - \tau' - N_i * T_p$	$T_i = D - N_i * N_e * T_p$	(5.9)

Variante (c) verdeutlicht, dass trotz periodischer Eingangsdatenströme, welche zum Teil sogar identische Periodendauern aufweisen, ein periodisches Systemverhalten nicht garantiert ist. Obwohl das Systemverhalten vorhersagbar und berechenbar ist, werden diese Fälle hier nicht betrachtet. Für alle Fälle, die nicht den Varianten (a) oder (b) zugeordnet werden können, sind die Konzepte für aperiodische Datenströme anzuwenden (siehe Kapitel 6).

Der gesamte Ablauf des Zustandstransfers lässt sich in Konfiguration und Durchführung unterteilen (Abbildung 5.2). Zur Konfiguration zählen die Identifikation von zu transferierenden Zustandswerten und die Festlegung des Transferverhaltens. Die Identifikation von Transferwerten beinhaltet zum einen die Auswahl von geeigneten Zustandspaaren in der Original- und Zielanfrage (Zustandspaarauswahl). Des Weiteren ist für die selektierten Zustände die Anzahl der notwendigen Werte so zu bestimmen, dass die Zustandstransferdauer minimal ist (Zustandsauswahl). Mit Kenntnis der Anzahl der Transferwerte über alle Zustände können die Reihenfolge und Anteile für die Übertragung der Zustände festgelegt werden. Reihenfolge und Anteile dienen zur Beschreibung des globalen Transferverhaltens. Ebenso ist die Transferreihenfolge innerhalb der ausgewählten Transfermengen festzulegen. Dieses lokale Transferverhalten wird durch Zustandstransferstrategien implementiert. Die Auswahl einer Zustandstransferstrategie kann die Zustandsauswahl und somit das globale Transferverhalten beeinflussen.

Basierend auf den Ergebnissen der Konfiguration findet die Durchführung statt. Dafür werden zunächst die Operatoren funktional erweitert, d. h. Operatoren, die Zustandspaare bilden, werden miteinander verbunden und entsprechend ihrer Rolle mit zusätzlichen Funktionen versehen. Danach beginnt der eigentliche Zustandstransfer, der insgesamt dem globalen Transferverhalten wie zuvor festgelegt folgt. Innerhalb der ausgewählten

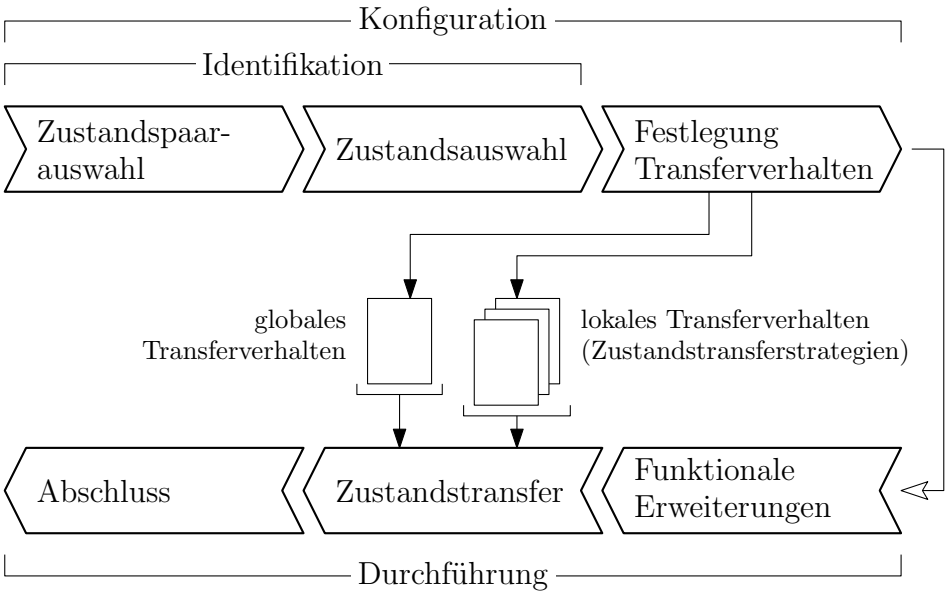


Abbildung 5.2: Ablauf des Zustandstransfers

Untermengen der Zustände werden die ausgewählten Zustandstransferstrategien genutzt. Nach Abschluss der Übertragung wird durch Entfernen der zusätzlichen Funktionen der Normalzustand der Zielanfrage hergestellt.

Im Verlauf des Zustandstransfers kommen unterschiedliche Methoden und Strategien zum Einsatz, welche in der nachfolgenden Übersicht mit ihrem jeweiligen Anwendungszweck aufgelistet sind.

- Zustandspaarauswahl – wird genutzt, um geeignete Paare von Zuständen für den Zustandstransfer zu bestimmen. Ein Paar besteht aus einem Zustandssender aus der Originalanfrage und einem Zustandsempfänger in der Zielanfrage. Das Ergebnis der Zustandspaarauswahl wird in einer Zustandspaartabelle gespeichert.

- Zustandsauswahl – dient zur Auswahl einer Transfermenge aus einem zu sendenden Zustand. Die Transfermenge enthält alle notwendigen Zustandswerte und hängt von verschiedenen Parametern der Anfragen und des Systems ab. Die jeweilige Anzahl der ausgewählten Werte wird ebenfalls in der Zustandspaartabelle abgespeichert und für die Konfiguration der Migration genutzt.

- Zustandstransferstrategie – legt das lokale Transferverhalten fest, d. h. die Übertragungsreihenfolge von Tupeln aus der Transfermenge. Die Zustandspaartabelle wird mit der Zustandstransferstrategie für jedes Zustandspaar ergänzt.

- Migrationsstrategie – setzt sich zusammen aus globalem und lokalen Transferverhalten. Im Allgemeinen werden dafür die Informationen aus der Zustandspaartabelle genutzt.

Die folgenden Abschnitte beschreiben Details des Zustandstransfers.

5.1 Festlegen von Zustandspaaren

Eine Voraussetzung für einen Zustandstransfer ist, dass Zustände der neuen Anfrage in der alten Anfrage existieren oder sich aus Zuständen der alten Anfrage erzeugen lassen. Für den ersten Fall müssen die entsprechenden Zustände in alter und neuer Anfrage schematisch und semantisch übereinstimmen. Zustandspaare können dann manuell oder durch automatische Suchverfahren festgelegt werden.

Von den bekannten Migrationsverfahren benutzen MS, HybMig und deren verallgemeinerte Verfahren, GMS und GHM, einen Zustandstransfer. Zum Festlegen der Zustandspaare wird dabei eine automatische Suche basierend auf den Zustandsbezeichnern benutzt. Die Anwendung dieser einfachen Methode zur Zustandspaarsuche ist möglich, da diese Migrationsverfahren auf Bäume angewendet werden, welche ausschließlich aus Verbundoperatoren bestehen, welche eine globalen zeitliche Fenstergröße besitzen. Die durchgeführten Änderungen führen immer zu semantisch äquivalenten Anfragen. Die Bezeichner repräsentieren das Schema der Tupel, z. B. Zustandsbezeichner $\mathcal{Z}_{ABC}$ für einen Zustand des Datenstroms $\mathcal{S}_{ABC}$, der Datentupel beinhaltet, welche aus Werten der Datenströme $\mathcal{S}_A$, $\mathcal{S}_B$ und $\mathcal{S}_C$ zusammengesetzt sind. Im Allgemeinen sind die Zustandsbezeichner für eine Zustandspaarsuche jedoch nicht ausreichend.

Sobald Änderungen durchgeführt werden, die eine semantische Äquivalenz der Zustände nicht garantieren, ist eine automatische Suche deutlich schwieriger zu realisieren. Beispiele für Verbundoperatoren sind die Änderung einer Fenstergröße oder des Verbundprädikats. Da bei diesen Modifikationen die Semantik verändert wird, jedoch nicht das Schema der Datentupel, wird die Änderung vom Zustandsbezeichner nicht widergespiegelt und dieser kann somit nicht zur Suche von äquivalenten Zuständen genutzt werden, zumindest nicht ausschließlich. Gleiches gilt, wenn Änderungen an den Grundfunktionen des Verbundoperators vorgenommen werden. Beispielsweise besteht die Möglichkeit, die Methoden zum Verwerfen von Zustandswerten oder zum Speichern der neuen Werte zu modifizieren, je nachdem, welche Werte zukünftig benötigt werden. Die Beispiele verdeutlichen, dass es bereits bei Bäumen von Verbundoperatoren Fälle gibt, in denen der Zustandsbezeichner allein nicht genügt, um ein äquivalentes Zustandspaar zu identifizieren. In solchen Fällen sind zusätzliche Informationen zum Vergleich der Semantik notwendig und jede durchgeführte Änderung muss berücksichtigt werden.

In heterogenen Anfragen, d. h. Anfragen, welche verschiedene Operatortypen enthalten, sind die beschriebenen Schwierigkeiten ebenso zu erwarten. Zudem treten weitere Situationen auf, die eine automatische Suche basierend auf einfachen Zustandsbezeichnern nicht möglich machen.

Für eine automatische Suche stellen Eingangsdatenströme, also Datenströme direkt von Datenquellen im DSMS, den einfachsten Fall dar. Da Eingangsströme unveränderte Werte darstellen und somit für die Original- und Zielanfrage identisch sind, sind auch ihre

Werte in den Zuständen der Operatoren semantisch äquivalent, wenn die Datenstromelemente ohne Filtern und ohne Modifikationen in den Zuständen gespeichert werden. Deshalb ist eine automatische Suche anhand der Zustandsbezeichner möglich. Hierbei können sogar Zustandspaare von verschiedenen Operatortypen in Frage kommen.

Für Ausgangsdatenströme von Operatoren ist eine automatische Suche basierend auf den Zustandsbezeichnern schwieriger. Vergleicht man Verbund- und Vereinigungsoperatoren, so verarbeiten beide jeweils zwei Eingangsdatenströme, z. B. $\mathcal{S}_A$ und $\mathcal{S}_B$, zu einem Ausgangsdatenstrom $\mathcal{S}_{AB}$. Die Semantik von $\mathcal{S}_{AB}$ unterscheidet sich für beide Fälle aber wesentlich und somit auch die inneren Zustände der nachfolgenden Operatoren. Während bei der Vereinigung die Datenstromwerte von $\mathcal{S}_A$ und $\mathcal{S}_B$ sequenziell als ein Datenstrom ausgegeben werden, erzeugt der Verbund Kombinationen der Datenstromwerte von $\mathcal{S}_A$ und $\mathcal{S}_B$ entsprechend des eingesetzten Verbundprädikats, gegebenenfalls auch mehrere Ergebnisse pro Eingangswert. Ein einfacher Zustandsbezeichner, wie bei MS oder HybMig, ist für eine automatische Suche in diesem Fall ungeeignet. Auch in diesem Beispiel werden zusätzliche Informationen zur Interpretation der Semantik benötigt.

Ebenso müssen die durchgeführten Änderungen für den Zustandstransfer und somit für die Zustandspaarsuche berücksichtigt werden. Wurde beispielsweise ein Filteroperator vor einen zustandsbehafteten Operator verschoben, dessen Zustand transferiert werden soll, muss auch der Zustand entsprechend gefiltert werden. Ansonsten werden ungültige Ergebnisse erzeugt, da durch einen ungefilterten Transfer Werte in den Zustand des neuen Operators gelangen, die als Eingangswerte zuvor gefiltert worden wären. Ein Filtern hinter dem Operator findet durch die Kompositionsänderung nicht mehr statt. Auch hierfür gilt, dass eine automatische Suche hilfreich aber keineswegs trivial zu realisieren ist.

Allgemein gilt, dass zwei Zustände ein Zustandspaar bilden, wenn sie semantisch äquivalent sind. Das schließt eine Äquivalenz der Schemen ein. Ist keine Zustandsäquivalenz gegeben, so besteht unter Umständen trotzdem die Möglichkeit, dass Zustandswerte für den neuen Zustand aus dem alten Zustand gewonnen werden können. Dies ist genau dann der Fall, wenn der neue Zustand eine Untermenge des alten Zustandes repräsentiert. Er kann dann durch Filtern aus dem alten Zustand extrahiert werden. Stimmen die Schemen beider Zustände nicht überein, können gegebenenfalls durch eine Transformation Werte für den neuen Zustand erzeugt werden. Können durch kein Verfahren Zustandswerte für den neuen Zustand gewonnen werden, dann gibt es kein Zustandspaar für diesen Zustand.

Die Umsetzung einer allgemeinen automatischen Suche von Zustandspaaren ist in den meisten Fällen nicht trivial und muss mehr Informationen als nur die Zustandsbezeichner mit einbeziehen. Für DSMS ist eine solche Lösung gegenwärtig nicht bekannt. In verwandten Forschungsfeldern, z. B. bei Datenbanken, gibt es Ansätze, die evtl. für die Entwicklung einer geeigneten Methode herangezogen werden können. So werden beispielsweise „Ontology Matching“ [WVV+01] oder „Graph Matching“ [SE05] genutzt, um semantische Ähnlichkeiten zu bestimmen. Es bleibt zu untersuchen, inwiefern sich derartige Ansätze auf Datenströme anwenden lassen.

Eine alternative Lösung ist das manuelle Festlegen von geeigneten Paarungen. Dafür ist jedoch eine Benutzerinteraktion notwendig. Zudem muss dem Nutzer die Semantik bekannt sein, was für die Migration von einfachen Anfragen noch vorstellbar ist. Inwiefern eine Migration von komplexen Anfragen realisierbar ist oder wie eine allgemeine automatische Suche auszusehen hat, bleibt zu untersuchen.

Zustandssender	Zustandsempfänger	Abhängigkeit
$Op_{(AB)}.\mathcal{Z}_A$	$Op_{(AB)}.\mathcal{Z}_A$	–
$Op_{(AB)}.\mathcal{Z}_B$	$Op_{(AB)}.\mathcal{Z}_B$	–
$Op_{((AB)C)}.\mathcal{Z}_C$	$Op_{(CD)}.\mathcal{Z}_C$	–
$Op_{(((AB)C)D)}.\mathcal{Z}_D$	$Op_{(CD)}.\mathcal{Z}_D$	–
$Op_{((AB)C)}.\mathcal{Z}_{AB}$	$Op_{((AB)(CD))}.\mathcal{Z}_{AB}$	$\mathcal{Z}_A$, $\mathcal{Z}_B$

Tabelle 5.1: Liste aller Zustandspaare für die Migration in Abbildung 5.7

Das Ergebnis einer Zustandspaarsuche, ob automatisch oder manuell, ist eine Liste aller Zustandspaare gespeichert in einer Zustandspaartabelle. Eine solche ist in Tabelle 5.1 für das Migrationsbeispiel in Abbildung 5.7 (S. 92) dargestellt. Für jedes Zustandspaar ist zudem die Abhängigkeit zu eventuellen Vorgängern in der Zielanfrage angegeben. Die Zustände $\mathcal{Z}_A$ bis $\mathcal{Z}_D$ beziehen ihre Daten direkt aus den Eingangsdatenströmen und sind daher unabhängig. Der Zustand $\mathcal{Z}_{AB}$ folgt in der Bearbeitung auf den Operator mit den Zuständen $\mathcal{Z}_A$ und $\mathcal{Z}_B$, sodass zwischen $\mathcal{Z}_{AB}$ und diesen Zuständen eine einseitige Abhängigkeit besteht. In den nächsten Schritten wird die Zustandspaartabelle mit weiteren Informationen ergänzt und schließlich zur Konfiguration der Migration benutzt.

5.2 Zustandsauswahl

Für jedes der identifizierten Zustandspaare ist anschließend die Menge der zu übertragenden Werte zu bestimmen. Für die Auswahl müssen verschiedene Eigenschaften der Zustände, der Operatoren, der gesamten Anfrage sowie der beteiligten, ausführenden Knoten berücksichtigt werden. Die ausgewählten Zustandswerte werden als Transfermenge bezeichnet.

Zustandseigenschaften Die Zustandseigenschaften sind im Wesentlichen an die Fenstereigenschaften gebunden, d. h. Typ, Größe und Verhalten des Fensters. Hinzu kommt die Tupelgröße. Die Fenstergrößen w_o und w_n beschreiben die Zustände in beiden Anfragen. Das Fenster des Zustandssenders ist definiert durch w_o, die originale Fenstergröße. Der Zustandsempfänger besitzt einen zu füllenden Zustand der Größe w_n, die neue Fenstergröße.

Bei Fenstertypen wird, wie in Abschnitt 2.2.4 beschrieben, zwischen anzahlbasierten und zeitbasierten Fenstern unterschieden. Betrachtet man einen einzelnen Operator in einem periodischen Datenstrom, dann sind beide Fenstertypen äquivalent. In Anfragen gilt diese Bedingung – auch in periodischen Datenströmen – nicht immer, und genau dann ist der Fenstertyp zu berücksichtigen.

Werden beispielsweise in einer Anfrage Verbundoperatoren verwendet, welche pro Eingangswert mehrere Ergebnisse produzieren, dann ist die Ergebnismenge eines Operators von seiner Ebene abhängig, da auf jeder Ebene eine unterschiedliche Anzahl von Eingangswerten auftritt. Zur Verdeutlichung wird die Originalanfrage der Verbundoperatoren aus Abbildung 3.1 (S. 31) verwendet. Zudem gelten nachfolgende Annahmen, um ein vollständiges Szenario zu erhalten. Alle vier Eingangsdatenströme haben die gleiche Periodendauer T und sind jeweils um die Zeit $T_v = T/4$ verschoben. Es werden gleitende zeitbasierte Fenster verwendet, deren Fenstergröße global $w = k * T$, $\forall k \in \mathbb{N}$ beträgt. Die Verbundoperation hat eine Selektivität von $sel = 1$, d. h. es werden keine Ergebnisse gefiltert, die Verbundoperation entspricht dem kartesischen Produkt[2]. Aus den Zuständen werden alle Werte verworfen, die Sub-Tupel enthalten, deren Zeitstempel außerhalb von w liegen. Es wird das Verhalten im stationären Zustand betrachtet.

Aufgrund von $sel = 1$ und $w = k * T$ werden für jeden Eingangswert k Ergebniswerte erzeugt. Dies gilt für jeden Operator und da die Ausgangswerte eines Operators die Eingangswerte seines Nachfolgers sind, erhöht sich die Anzahl der gesamten Werte mit jeder Ebene. So erhält $Op_{((AB)C)}$ von seinem Vorgänger k Werte für jeden Datenquellenwert. Sein Zustand $\mathcal{Z}_{AB}$ enthält somit k^2 Werte. Zustand $\mathcal{Z}_{ABC}$ im Operator $Op_{(((AB)C)D)}$ beinhaltet k^3 Werte. In diesem Szenario erhält die Datensenke für jeden Eingangswert, egal von welcher Datenquelle, k^3 Ergebnisse. Würden anzahlbasierte Fenster mit einer globalen Fenstergröße in diesem Szenario verwendet, könnten in den Zuständen $\mathcal{Z}_{AB}$ und $\mathcal{Z}_{ABC}$ nicht alle Werte gespeichert werden und es würden weniger Ergebnisse erzeugt.

Daraus folgt, dass bei der Verwendung von zeitbasierten Fenstern die Anzahl der beinhalteten Tupel unter Berücksichtigung der Komposition und Konfiguration der Anfrage berechnet werden muss. Für den Zustandstransfer ist dann gegebenenfalls eine größere Transfermenge relevant. Bei anzahlbasierten Fenstern ist keine Berechnung notwendig, da dort die Anzahl direkt von der Fenstergröße verwendet werden kann.

Eine weitere Eigenschaft ist die Schrittweite δ eines Fensters. Bei $\delta > 1$ ist zu überprüfen, ob ein Zustandstransfer überhaupt notwendig ist oder auf welche Werte bei der Übertragung verzichtet werden kann. Werden z. B. springende Fenster benutzt, und die Zustandstransferzeit überschreitet die Zeit bis zum nächsten Zurücksetzen eines Fensters, kann auf einen Zustandstransfer verzichtet werden. Die Migration kann dann durch Umschalten zur neuen Anfrage zum Rücksetzzeitpunkt sehr einfach realisiert werden.

Schließlich ist die Speichergröße eines Datentupels eine entscheidende Eigenschaft. Unter Berücksichtigung der Systemeigenschaften kann daraus die Tupeltransferzeit T_t errechnet werden.

[2]Ein Eingangswert wird mit jedem Zustandswert verbunden und als Ergebnis ausgegeben

Operatoreigenschaften Eine Operatoreigenschaft, die den Zustandstransfer wesentlich beeinflusst, ist die Verarbeitungszeit T_p. Sie ist zum einen vom eingesetzten Algorithmus und den zu verarbeitenden Daten abhängig, aber auch von den Systemeigenschaften, z. B. der CPU-Leistung. Sie kann aber ohnehin nur zur Laufzeit bestimmt werden. Dafür muss der Operator beobachtbar sein. Existierende DSMS, beispielsweise Borealis [AAB+05], haben dafür geeignete Schnittstellen integriert. Eine weitere Möglichkeit ist die Bestimmung der Verarbeitungszeit durch Laufzeittests in einer geeigneten Testumgebung. Dieses Verfahren liefert aber nur genäherte Werte, da es nicht den aktuellen Systemzustand abbildet. Mit Kenntnis der Verarbeitungszeit kann die Pausendauer errechnet werden, also die Zeit, in der der Operator keine Tätigkeiten durchführt. Zudem kann die Verarbeitungszeit für Vergleiche herangezogen werden. Gegebenenfalls ist es günstiger, zusammengesetzte Tupel im neuen Operator zu berechnen, statt zu transferieren. Das ist beispielsweise genau dann der Fall, wenn $T_p < T_t$ und wenn alle notwendigen Basistupel für die Berechnung des Zustandes bereits zur Verfügung stehen.

Außerdem ist für einen Operator die Anzahl der eingehenden Werte pro Zyklus (N_i) zu beachten. Die Anzahl lässt sich aus dem Verhalten der vorgeschalteten Teilanfrage bestimmen, d. h. wie viele Ergebnisse produziert diese Teilanfrage für einen Eingangswert. Eigenschaften wie die Selektivität und die Position (Ebene) eines Operators in der Anfrage wirken sich auf das Ausgabeverhalten von Operatoren aus und dadurch auf N_i der Nachfolgeoperatoren. Es werden hier ausschließlich Fälle mit einer konstanten Anzahl von N_i berücksichtigt. Ist eine variable Anzahl von Eingangswerten zu erwarten, ist das Transferverhalten nicht exakt vorhersagbar und die Konzepte für aperiodische Datenströme werden angewendet.

Anfrageeigenschaften Eigenschaften der Anfrage, welche die Konfiguration der Migration beeinflussen, sind das Verhalten der Eingangsströme und der Aufbau der Anfrage. Die Charakteristik der Eingangsdatenströme und entsprechende Konsequenzen wurden bereits zu Beginn dieses Kapitels beschrieben. Zu ergänzen ist, dass bei der Verwendung von schwankungsbeschränkten Datenströmen deren Schwankung zu berücksichtigen ist. Für die Berechnung der Pausendauer ist der Mindestabstand D [Ham05] zu verwenden. Die Bestimmung von D kann, ähnlich wie bei T_p, durch Beobachtung des Systems oder durch vorheriges Festlegen erfolgen.

Beim Aufbau der Anfrage ist neben den Positionen und Eigenschaften der Operatoren, deren Einflüsse bereits beschrieben wurden, die Anzahl und Charakteristik der Zustände interessant. Die Zustände der Zielanfrage lassen sich unterteilen in (a) Zustände, die durch Zustandstransfer aufgefüllt werden können, (b) Zustände, die durch Berechnung auf dem neuen Verarbeitungsknoten aufgefüllt werden können (ähnlich BSC [YKPS07]) und (c) Zustände die nicht durch Zustandstransfer oder Berechnung aufgefüllt werden können. Insbesondere Zustände der letzten Kategorie können einen Zustandstransfer überflüssig machen, beispielsweise wenn deren Anlaufzeit die Zustandstransferzeit der Anfrage übersteigt.

Systemeigenschaften Abschließend werden die Eigenschaften der Bearbeitungsknoten und des Gesamtsystems betrachtet. Der Einfluss der CPU-Leistung auf die Verarbeitungszeit wurde bereits beschrieben. Wird der Zustand zwischen verteilten Bearbeitungsknoten transferiert, ist die entsprechende Netzwerkverbindung zu berücksichtigen. Diese kann durch eine Datenübertragungsrate und eine Latenzzeit charakterisiert werden. Zusammen mit der Tupelgröße lässt sich daraus die Tupeltransferzeit T_t errechnen, d. h. die Zeit, die benötigt wird, um ein Tupel vom Zustandssender zum Zustandsempfänger zu übertragen.

Formelzeichen	Beschreibung
D	Mindestabstand eines SBDS
T	Periodendauer
T_p	Verarbeitungszeit eines Eingangswertes
w_o	originale Fenstergröße
w_n	neue Fenstergröße
N_i	Anzahl Eingangswerte pro Zyklus
T_t	Tupeltransferzeit

Tabelle 5.2: Ausgangsgrößen für die Berechnung der Migrationsparameter

Zusammenfassend werden alle Eigenschaften klassifiziert. Ihre Auswirkungen auf die Berechnung der Migrations- und Zustandstransferparameter werden in den nächsten Unterabschnitten im Detail diskutiert. Alle numerischen Eigenschaften sind in Tabelle 5.2 aufgelistet. Diese sind die Ausgangsgrößen für die Berechnung der Migrationsparameter. In periodischen Datenströmen werden T und T_p zur Berechnung der Pausendauer genutzt. Für schwankungsbeschränkte Datenströme (SBDS) wird statt T der Mindestabstand D verwendet. Alle übrigen Parameter der Tabelle gelten unabhängig vom Datenstromtyp. Weitere Eigenschaften lassen sich auf numerische Eigenschaften abbilden, z. B. CPU-Leistung auf T_p oder Netzwerkeigenschaften auf T_t. Außerdem gibt es Eigenschaften, die sich nicht numerisch abbilden lassen, sich aber trotzdem auf die Berechnung und auf die Migrationskonfiguration auswirken. Dazu zählen die Fenstertypen, bestimmte Eigenschaften der Kompositionen der Original- und Zielanfrage sowie die Klassen der Zustände, insbesondere Zustände, die nicht durch Zustandstransfer aufgefüllt werden können.

Da die Berechnung vor der Durchführung der Migration ausgeführt wird und zur Laufzeit keine dynamischen Änderungen vorgesehen sind, werden nachfolgende Vereinfachungen angenommen. Die Zeit, die zum Steuern des Migrationsablaufs notwendig ist, wird vernachlässigt oder ist in T_p enthalten. Dazu zählen z. B. die Auswahl der nächsten Aktivität, das Aufrufen von Operatoren, die Werteauswahl aus dem Zustand im Zustandssender, das Einsortieren im Zustand des Zustandsempfängers oder das Protokollieren von Aktivitäten. Die Tupeltransferzeit T_t wird als Abstraktion für den Transfer

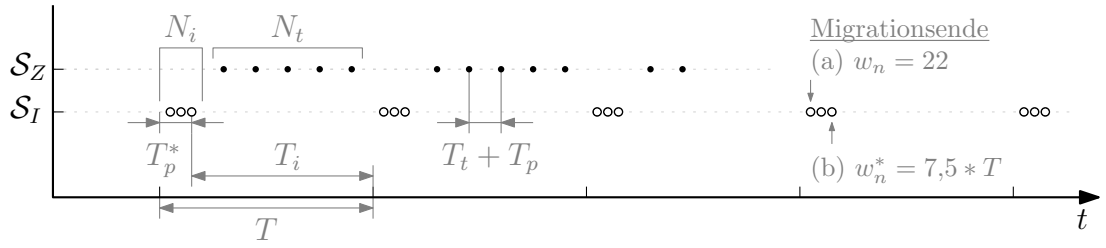


Abbildung 5.3: Beispielhaftes Schema für den Zustandstransfer eines einzelnen Fensters mit $N_i = 3$, $N_t = 5$

verwendet, d. h. Netzwerkparameter oder Datentupelgrößen werden nicht betrachtet.

In den nächsten Unterabschnitten wird die Berechnung für Anfragen mit einem Zustand, mit mehreren homogenen Zuständen und mit mehreren heterogenen Zuständen gezeigt, wobei verschiedene Fallbeispiele zur Erläuterung genutzt werden.

5.2.1 Berechnung für Anfragen mit einem Zustand

Das einfachste Szenario einer Migration mit Zustandstransfer ist die Migration einer Anfrage mit einem Zustand. In diesem Fall entspricht das lokale Transferverhalten auch dem globalen Transferverhalten. Ein beispielhaftes Schema für den Zustandstransfer eines einzelnen Fensters ist in Abbildung 5.3 dargestellt. Darin sind die Datenwerte des Eingangsstroms $\mathcal{S}_I$ und die Werte des Zustandstransferdatenstroms $\mathcal{S}_Z$ in ihrer zeitlichen Reihenfolge abgebildet. Jede Marke kennzeichnet das Ende der Bearbeitung eines einzelnen Datenstromwertes durch den Zustandsempfänger.

Das Berechnungsmodell ist so ausgelegt, dass die Zeit, die pro Pause für den Zustandstransfer verwendet wird, immer kleiner ist als die verfügbare Pausendauer T_i. Damit wird vermieden, dass sich die Anfragen im Zustandstransfer befinden, wenn ein neuer Eingangswert auftritt[3]. Die Pausendauer ist in periodischen Datenströmen die Differenz zwischen Periodendauer und Verarbeitungszeit $T_i = T - T_p^*$ wobei T_p^* die Verarbeitungszeit aller Eingangswerte eines Zyklus beschreibt. Sie errechnet sich aus der Anzahl der Eingangswerte pro Zyklus N_i und der Verarbeitungszeit eines einzelnen Eingangswertes T_p, so dass $T_p^* = N_i * T_p$.

In schwankungsbeschränkten Datenströmen ist die Pausendauer die Differenz aus Mindestabstand D und der Verarbeitungszeit, $T_i = D - T_p^*$. Als weitere Bedingung gilt, dass mindestens ein Zustandswert pro Pause übertragen werden kann, d. h. $T_t < T_i$. Es besteht zwar grundsätzlich die Möglichkeit, Werte zu fragmentieren, dieser Fall wird hier jedoch nicht betrachtet.

Zunächst wird die Anzahl der Zustandswerte berechnet, die während einer Pause übertragen werden können. Relevant dafür ist die zur Verfügung stehende Zeit $(1-r) * T_i$ mit

[3]Diese Annahme wird im nächsten Kapitel aufgelöst und es wird eine Lösung präsentiert, die prinzipiell auch in periodischen Datenströmen anwendbar ist.

dem Reserveanteil r ($0 \leq r < 1$). Die Reserve ist optional und dient der Gewährleistung einer Zuverlässigkeit, z. B. um unerwartete Schwankungen zu kompensieren. Wird keine Reserve benötigt, nutzt man $r = 0$. Die Summe $T_t + T_p$ entspricht der Zeit, die für die Übertragung und die Verarbeitung eines Datentupels aus $\mathcal{S}_Z$ benötigt wird. Für T_p ist die Verarbeitungszeit in der Zielanfrage zu verwenden. Falls keine Berechnung und keine Sortierung benötigt werden, ist T_p unter Umständen so klein, dass diese Zeit vernachlässigt werden kann, beispielsweise wenn der Zustand in der Zielanfrage nur aufgefüllt wird und keine Berechnung von Zwischentupeln notwendig ist. Das Gleiche gilt für T_t, wenn z. B. der Transfer lokal durchgeführt wird, etwa durch die Nutzung gemeinsamer Zustände[4]. Durch die Verwendung der Abrundungsfunktion erhält man N_t, die Anzahl übertragbarer Zustandswerte während einer Pause.

$$N_t = \left\lfloor \frac{(1 - r) * T_i}{T_t + T_p} \right\rfloor \tag{5.10}$$

Als nächstes kann die Anzahl der Zyklen N_c berechnet werden, die notwendig sind, um das neue Fenster vollständig mit Werten zu füllen. Im Falle eines zeitbasierten Fensters mit der Fenstergröße w_n^* muss diese zuvor in eine Anzahl umgerechnet werden. Mit Hilfe von N_i kann beispielsweise die Position eines Operators in der Anfrage berücksichtigt werden. Die daraus folgende Umrechnung der zeitbasierten Fenstergröße in die entsprechende Anzahl lautet

$$w_n = \left\lceil \frac{w_n^*}{T} \right\rceil * N_i. \tag{5.11}$$

Für andere Anwendungsfälle ist diese Formel gegebenenfalls anzupassen. Für einen periodischen Datenstrom mit $w_n^* = k * T$ ist es unter Umständen ungünstig, ganzzahlige Werte für k zu nutzen, da dadurch die untere Grenze des Fensters in einen Zeitabschnitt mit vergangenen Eingangswerten fällt. Je nach Schwankung können daraus unterschiedlich große Mengen von Datenwerten für die Berechnung resultieren. Für die weitere Erklärung wird die Verwendung eines geeigneten k ($k \in \mathbb{R},\ k > 0$) angenommen.

Für die Berechnung von N_c werden sowohl Eingangswerte ($\mathcal{S}_I$) als auch Werte aus dem Zustandstransfer ($\mathcal{S}_Z$) berücksichtigt. Neben der neuen Fenstergröße w_n werden die Anzahl der Transferwerte N_t und die Anzahl der Eingangswerte pro Zyklus N_i benötigt. Der Zustandstransfer beginnt mit der ersten vollständigen Pause direkt nachdem[5] die ersten Eingangswerte[6] verarbeitet wurden. Im neuen Fenster befinden sich dann bereits N_i Werte. Es sind weitere $w_n - N_i$ Werte notwendig, um das Fenster zu vervollständigen.

[4] Im weiteren Verlauf der Arbeit wird immer die vollständige Variante ($T_t + T_p$) genutzt. Diese ist bei Bedarf zu vereinfachen. Eine Division durch 0 ist jedoch zu vermeiden. Die Vereinfachung ist auch bei den nachfolgenden Berechnungen zu berücksichtigen, z. B. bei der Zustandstransferdauer.

[5] Gegebenenfalls ist auch die Zeit bis zum Eintreffen des ersten neuen Wertes für den Zustandstransfer nutzbar. Eine mögliche Erweiterung wird später diskutiert.

[6] „Erste Eingangswerte“ bezieht sich auf die Migration, d. h. es sind die ersten Eingangswerte während der Migration gemeint.

In jedem weiteren Zyklus werden maximal N_i neue Werte und N_t Transferwerte in das Fenster eingefügt. Mittels Aufrundungsfunktion kann N_c, die Anzahl der notwendigen Transferzyklen, errechnet werden.

$$N_c = \left\lceil \frac{w_n - N_i}{N_t + N_i} \right\rceil \tag{5.12}$$

Schließlich lassen sich die Anzahl neuer und die Anzahl alter Tupel (N_n und N_o) im neuen Fenster bei Migrationsende berechnen, d. h. wie viele alte und neue Datenstromwerte beinhaltet das Fenster, wenn das erste Ergebnis vom neuen Operator ausgegeben wird. Wesentlich dafür ist, unter welchen Bedingungen Ergebnisse vom Operator errechnet und ausgegeben werden. Die Berechnung wird für zwei Beispiele beschrieben. Für andere Fälle ist gegebenenfalls eine konfigurationsspezifische Berechnung durchzuführen.

Im ersten Fall (a) wird für jeden Eingangswert ein Ergebnis erzeugt ($sel = 1$). Dies ist beispielsweise bei anzahlbasierten, gleitenden Fenstern der Fall. Da jeder Wert des Eingangsdatenstroms das Fenster vervollständigen kann und somit das Migrationsende und das Umschalten zum neuen Operator auslöst, ist zunächst zu berechnen, wie viele neue Werte im letzten Zyklus für die Vervollständigung des Fensters sorgen. Mit Hilfe der Modulo-Funktion kann die Hilfsgröße N_r bestimmt werden, die anschließend bei einer Fallunterscheidung als Entscheidungskriterium dient.

$$N_r = (w_n \mod (N_i + N_t)) \tag{5.13}$$

$$N_n = N_i * N_c + \begin{cases} N_r, & \text{falls } 0 < N_r \leq N_i \\ 1, & \text{sonst} \end{cases} \tag{5.14}$$

Im zweiten Fall (b) wird in jedem Zyklus ein Ergebnis erzeugt, wenn alle N_i Eingangswerte im Fenster gespeichert wurden. Die Fenstergröße ist dann immer ein Vielfaches von N_i. Eine Berechnung der Hilfsgröße N_r ist deshalb nicht notwendig, da $N_r = N_i$ ist. Deshalb kann die Berechnung von N_n vereinfacht werden zu

$$N_n = N_i * (N_c + 1). \tag{5.15}$$

In jedem Fall gilt, dass sich die Anzahl der notwendigen alten Werte, also die Größe der Transfermenge, errechnet aus

$$N_o = w_n - N_n. \tag{5.16}$$

Die Größe N_o wird hier unabhängig von den tatsächlich zur Verfügung stehenden Werten betrachtet. Die minimale Originalfenstergröße zur Gewährleistung von N_o Transferwerten ist jedoch von der Zustandstransferstrategie abhängig, weshalb die Betrachtung der Anzahl der Transferwerte bei der Beschreibung der Zustandstransferstrategien fortgeführt wird. Generell gilt, sind zu wenig Werte im alten Fenster verfügbar, muss die Anzahl der fehlenden Werte mit neuen Werten kompensiert werden. Dadurch verlängert sich die Migrationsdauer.

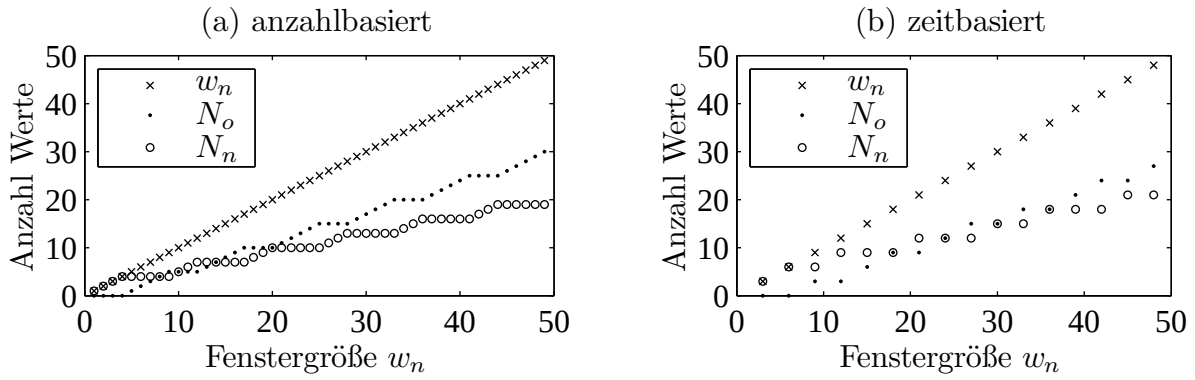


Abbildung 5.4: Zusammensetzung eines neuen Fensters aus neuen und alten Datenstromwerten für zwei beispielhafte Fälle mit $N_i = 3$ und $N_t = 5$

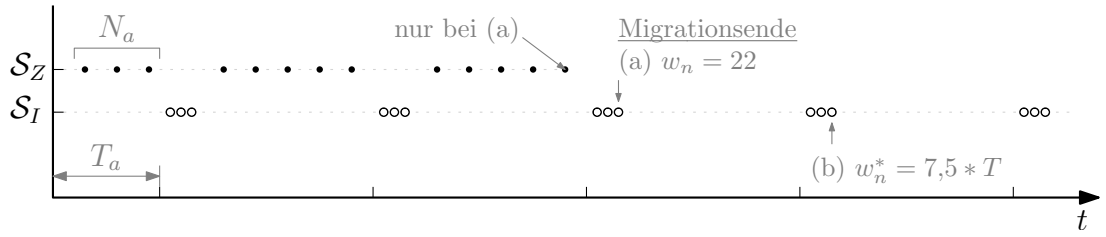


Abbildung 5.5: Verhalten bei Verwendung von sofortigem Zustandstransfer für ein einzelnes Fenster mit $N_i = 3$, $N_t = 5$

In Abbildung 5.4 ist die Zusammensetzung eines neuen Fensters für eine Beispielkonfiguration mit $N_i = 3$ und $N_t = 5$ dargestellt. Die Diagramme zeigen, wie sich für die beschriebenen Fälle (a) und (b) die Anteile von N_n und N_o für verschiedene Fenstergrößen entwickeln. Ebenso ist in Abbildung 5.3 das Migrationsende für Beispiele der beiden Fälle markiert. Die Fenstergröße $w_n^* = 7{,}5 * T$ von Fall (b) entspricht dabei 24 Werten, welche bei Migrationsende aus jeweils 12 neuen und alten Werten zusammengesetzt sind.

Abschließend folgen Anmerkungen zu Sonderfällen und Erweiterungsmöglichkeiten. Der wahrscheinlich am häufigsten auftretende Fall bei gleitenden Fenstern ist ein einzelner Eingangswert pro Zyklus, d. h. $N_i = 1$. Dafür gelten alle Formeln unverändert. Die Formel zur Berechnung der Anzahl der neuen Werte (5.14) kann vereinfacht werden zu

$$N_n = N_c + 1. \tag{5.17}$$

Die Fälle (a) und (b) zeigen, dass sich für $N_i > 1$ die Art und Weise der Ergebnisausgabe auf das Verhältnis von neuen und alten Werten im neuen Fenster auswirkt. Das bedeutet, dass für unterschiedliche Konfigurationen der Ergebnisausgabe auch verschiedene Formeln für die Berechnung von N_n aus dem Ausgabeverhalten abzuleiten sind. Unter Umständen ist eine Hilfsgröße wie N_r zu verwenden.

Sofortiger Zustandstransfer Im bisherigen Verlauf wurde mit dem Zustandstransfer nach der Verarbeitung der ersten N_i Eingangswerte begonnen. Der Ansatz lässt sich erweitern, indem die Zeit bis zum Eintreffen der ersten Eingangswerte für den Zustandstransfer genutzt wird. Ein sofortiger Beginn des Zustandstransfers kann unter Umständen die Migration um eine Periode verkürzen oder auch zur Erhöhung der Reserve in zukünftigen Übertragungszyklen genutzt werden, da dann weniger Werte pro Pause übertragen werden müssen. Zur Realisierung muss die Zeit bis zum Eintreffen der ersten Eingangswerte abschätzbar sein, z. B. durch die Berechnung der Differenz aus aktueller Zeit und erwartetem Ankunftszeitpunkt der nächsten Eingangswerte. In Abbildung 5.5 ist diese Zeit als T_a eingetragen. Im Unterschied zu den anderen Berechnungen kann dieser Wert ausschließlich zur Laufzeit bestimmt werden. Dafür ist eine Methode zur Vorhersage des erwarteten Ankunftszeitpunktes der nächsten Eingangswerte erforderlich. Geht man davon aus, dass in einem System T_a zur Laufzeit berechenbar ist, kann daraus die Anzahl der Transferwerte errechnet werden, die bis zum Eintreffen der ersten Eingangswerte übertragen werden können (N_a). Wie bei der Berechnung von N_t, kann auch für N_a ein Reserveanteil berücksichtigt werden. Mit $r * T_i$ ist dieser ebenso groß wie während einer regulären Pause.

$$N_a = \left\lfloor \frac{T_a - r * T_i}{T_t + T_p} \right\rfloor \tag{5.18}$$

Die Formeln zur Konfiguration des Zustandstransfers sind dann unter Berücksichtigung von N_a anzupassen. Für die Berechnung von N_c und N_r (Gleichungen (5.12) und (5.13)) ist w_n mit $w_n - N_a$ zu ersetzen, da entsprechend weniger Werte nach der Verarbeitung der ersten Eingangswerte übertragen werden müssen. Dies wirkt sich folglich auch auf die Berechnung von N_n aus.

Betrachtet man die bereits diskutierten Fälle (a) und (b), so lassen sich unterschiedliche Effekte der sofortigen Zustandsübertragung beobachten. Beide Fälle sind in Abbildung 5.5 für $N_a = 3$ dargestellt. Bei Fall (a) endet die Migration knapp eine Periode ($T - 2 * T_p$) früher als ohne sofortigen Zustandstransfer. Insgesamt werden nun 13 Zustandswerte übertragen und die Anzahl der verwendeten neuen Werte verringert sich dementsprechend um einen Wert.

Für den Fall (b) kann keine Verbesserung bezüglich der Migrationszeit erzielt werden. Dies wird bereits bei der Berechnung von N_c deutlich, die für die Ausführung mit und ohne sofortigen Zustandstransfer das gleiche Ergebnis ($N_c = 3$) liefert. Die Ursache ist, dass N_n und N_o für diesen Fall immer Vielfache von N_i sind und T_a nicht ausreicht, um die fünf Werte (drei neue und zwei Zustandstransfer), welche zwischen letzter und vorletzter Periode des Transfers verarbeitet werden, zu übertragen. Die Abbildung 5.6 enthält weitere Beispiele mit verschiedenen Fenstergrößen zur Verdeutlichung der Problematik. Die potentiellen Endtupel einer schnelleren Migration sind jeweils mit einem Kreuz markiert. Zudem sind jeweils die Werte für N_a^* und N_a' angegeben.

Die Größe N_a' dient als Vergleichsgröße für N_a, um festzustellen, ob eine schnellere

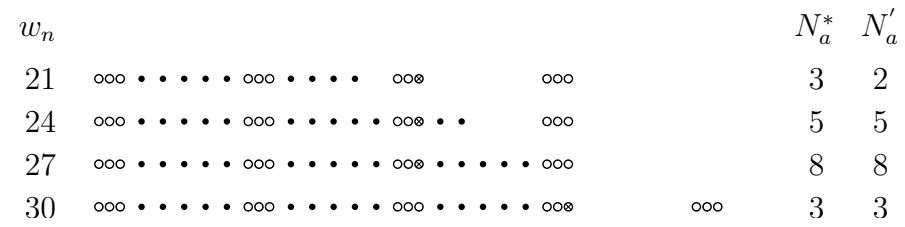


Abbildung 5.6: Beispiele zur Darstellung der Anwendbarkeit eines sofortigen Zustandstransfers für Fall (b) mit $N_i = 3$, $N_t = 5$ (Symbole entsprechend Abbildung 5.5)

Migration möglich ist. Das ist genau dann der Fall, wenn $N_a \geq N_a'$. Die Berechnung von N_a' erfolgt mit Hilfe von N_a^*, der Anzahl der Werte, die während der letzten Periode der Migration auftreten. Bei der Berechnung beider Größen wird teilweise N_o^* benutzt. Das ist die Anzahl der übertragenen Zustandswerte während der letzten Pause mit Zustandstransfer. Für die Vergleichsgröße N_a^* ist N_o^* mit $N_a = 0$ zu verwenden, was N_o^* ohne Verwendung des sofortigen Zustandstransfers entspricht.

$$N_o^* = ((N_o - N_a - 1) \mod N_t) + 1 \tag{5.19}$$

Für Fall (a) lautet die Berechnung von N_a^*

$$N_a^* = N_r + \begin{cases} N_o^*\,(N_a{=}0)\,, & \text{falls } \lceil N_o/N_t \rceil = \lceil N_n/N_i \rceil - 1 \\ 0, & \text{sonst.} \end{cases} \tag{5.20}$$

Für Fall (b) sieht die Berechnung von N_a^* ähnlich aus.

$$N_a^* = N_i + \begin{cases} N_o^*\,(N_a{=}0)\,, & \text{falls } \lceil N_o/N_t \rceil = \lceil N_n/N_i \rceil - 1 \\ 0, & \text{sonst.} \end{cases} \tag{5.21}$$

Es wird jeweils überprüft, ob in der letzten Pause der Migration Zustandswerte übertragen werden. Deren Anzahl wird dann zu den letzten neuen Werten addiert. Für beide Fälle kann mit N_a^* die Vergleichsgröße N_a' erzeugt werden. Bei Fall (a) sind N_a^* und N_a' identisch.

$$N_a' = N_a^* \tag{5.22}$$

Bei Fall (b) ist eine Besonderheit zu berücksichtigen, die sich aus der Tatsache ergibt, dass N_n und N_o immer Vielfache von N_i sind. Für manche Konfigurationen kann es sein, dass der Zustandstransfer in der vorletzten Pause der Migration endet, d. h. die letzte Pause wird nicht für den Zustandstransfer genutzt und die vorletzte nur teilweise. Dadurch sind unter Umständen zusätzliche Plätze für die Zustandsübertragung in der vorletzten Pause vorhanden. In Abbildung 5.6 trifft das für $w_n = 21$ zu, da hier ein weiterer Wert übertragen werden kann. Bei der Berechnung von N_a' ist dieser Umstand zu beachten. Für Fall (b) wird N_a' wie folgt berechnet.

$$N'_a = \begin{cases} N^*_a, & \text{falls } N^*_a > N_i \\ N^*_a - (N_t - N^*_o), & \text{sonst} \end{cases} \tag{5.23}$$

Wie bereits genannt, wird ein früheres Migrationsende erreicht, wenn $N_a \geq N'_a$. Im Beispiel (Abbildung 5.6) mit $N_a = 3$ trifft das neben $w_n = 21$ auch auf $w_n = 30$ zu. Mit $N_a = 5$ könnte die Migration auch für $w_n = 24$ eher enden. Für die Fenstergröße $w_n = 27$ ist keine schnellere Migration möglich, da gilt $N_a \leq N_t$ in diesem Fall jedoch $N'_a > N_t$ ist.

Die Beispiele verdeutlichen, dass durch die Verwendung eines sofortigen Zustandstransfers ein schnelleres Migrationsende möglich ist. Das trifft jedoch nicht auf jede Konfiguration zu. Unabhängig davon kann der sofortige Zustandstransfer immer benutzt werden, um die zu übertragenden Zustandswerte auf die Pausen zu verteilen und dadurch gegebenenfalls größere Reserveanteile zu schaffen. Da die Parameter für den sofortigen Zustandstransfer ausschließlich zur Laufzeit bestimmt werden können, sind auch alle nachfolgenden Berechnungen zur Festlegung des Transferverhaltens zur Laufzeit auszuführen.

5.2.2 Berechnung für Anfragen mit mehreren Zuständen

Für die Anwendung des Berechnungsmodells auf Anfragen mit mehreren Zuständen sei wiederholt, dass das Verhalten der Anfrage in jedem Fall dem anfangs beschriebenen periodischen Verhalten der Varianten (a) oder (b) (Abbildung 5.1) entsprechen muss.

Zudem ist zwischen homogenen und heterogenen Zuständen zu differenzieren. Die Unterscheidung bezieht sich auf alle Zustände, die für den Zustandstransfer ausgewählt wurden, da alle weiteren Zustände innerhalb der Anfrage durch Berechnung gefüllt werden. Zustände sind homogen, wenn sie in allen Eigenschaften vollständig übereinstimmen, ansonsten sind sie heterogen. Unterscheidungskriterien sind alle zuvor beschriebenen Zustands-, Operator- und Anfrageeigenschaften. Hinzu kommen Eigenschaften wie T_p und T_t. Geringe Abweichungen bei T_p und T_t können mit der Anwendung des Berechnungsmodells für schwankungsbeschränkte periodische Datenströme bzw. mit der Verwendung der Maximalwerte kompensiert werden.

Mit den Anfrageeigenschaften wurden Eigenschaften von Zuständen genannt, welche unter Umständen dafür sorgen, dass die Menge der Zustandspaare reduziert werden kann. Es wurde zwischen Zuständen unterschieden, deren Auffüllen erfolgt durch (a) Zustandstransfer, (b) durch Berechnung in der Zielanfrage und (c) weder durch Zustandstransfer noch durch Berechnung. Die Zustände der Kategorie (a) sind alle diejenigen, die durch die Zustandspaarauswahl identifiziert wurden. Sollten sich darunter Zustände befinden, die auch zur Kategorie (b) gehören, also berechenbare Zustände, dann ist zu überprüfen, ob es günstiger ist, einen solchen Zustand in der Zielanfrage zu berechnen oder einen Zustandstransfer durchzuführen. Eine Berechnung ist günstiger, falls alle Eingangswerte eines Zyklus schneller berechnet als transferiert werden können.

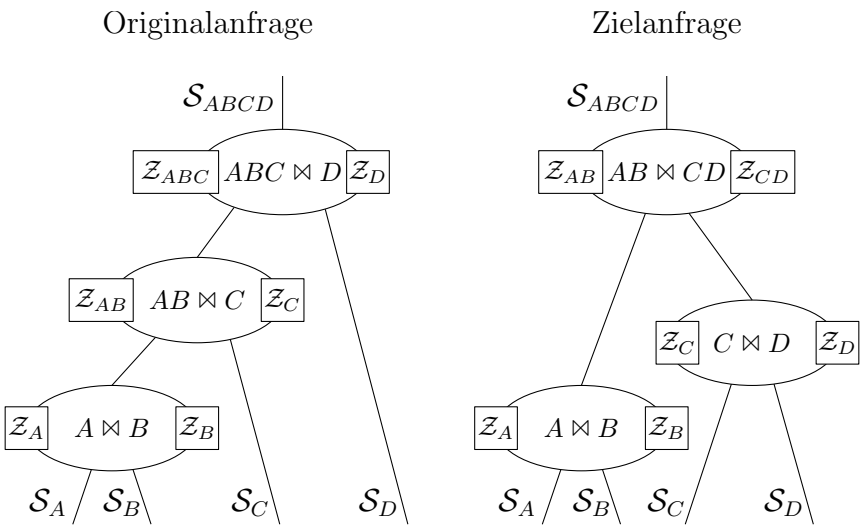


Abbildung 5.7: Migrationsbeispiel

Die Zustandspaarmenge kann um diese Zustände reduziert werden. Ein möglicher Fall eines Zustands der Kategorie (c) ist, wenn die Zielanfrage eine Datenquelle enthält, die nicht in der Originalanfrage enthalten war. Ein zustandsbehafteter Operator, der Daten dieser Datenquelle verarbeitet, besitzt somit einen Zustand, der sich weder transferieren noch errechnen lässt.

Zustände der Kategorie (c) können einen Zustandstransfer sogar unnötig machen, z. B. unter der Bedingung, dass ein Wechsel zur neuen Anfrage erst im stationären Zustand durchgeführt wird, d. h. alle Zustände der neuen Anfrage müssen vollständig gefüllt sein. Existiert in der neuen Anfrage mindestens ein Zustand der Kategorie (c), ist die Migrationszeit mindestens so groß wie dessen Anlaufzeit, sind es mehrere, dann zählt die maximale dieser Anlaufzeiten. Ist diese zudem die maximale Anlaufzeit aller zustandsbehafteten Operatoren der Zielanfrage, dann erübrigt sich eine Migration mit Zustandstransfer, und die neue Anfrage kann beispielsweise durch die Strategie PT in Betrieb genommen werden. Die Zustandspaarmenge kann dann verworfen werden, da eine andere Migrationsstrategie verwendet wird.

Im Migrationsbeispiel in Abbildung 5.7 gehören die Zustände $\mathcal{Z}_A$, $\mathcal{Z}_B$, $\mathcal{Z}_C$, $\mathcal{Z}_D$ und $\mathcal{Z}_{AB}$ zur Kategorie (a). Der Zustand $\mathcal{Z}_{AB}$ gehört außerdem zur Kategorie (b), er kann berechnet oder transferiert werden und ist entsprechend für die Migration zu bewerten. Ein Zustand der Kategorie (c) ist im Migrationsbeispiel nicht enthalten. Die Zustände $\mathcal{Z}_{CD}$ und $\mathcal{Z}_{ABC}$ gehören zwar auch zur Kategorie (b), haben aber keinen Einfluss auf die Zustandspaarmenge, da sie nicht der Kategorie (a) zugeordnet werden können. Nach der Reduktion der Zustandspaarmenge sind die verbleibenden Zustände bezüglich ihrer Homogenität zu bewerten. Die Größe der Zustandspaarmenge ist charakterisiert mit N_z, der Anzahl der Zustandspaare für den Zustandstransfer.

Homogene Zustände

Homogene Zustände stimmen in allen Eigenschaften überein und sind unabhängig voneinander. Der Zustandstransfer von homogenen Zuständen lässt sich als Transfer eines einzelnen Zustandes interpretieren und mit dem Modell des vorherigen Abschnittes berechnen. Zu beachten sind jedoch die Anzahl der Eingangsdatenströme (N_e) und deren Reihenfolge, weshalb geringe Anpassungen des Modells erforderlich sind.

Im Migrationsszenario in Abbildung 5.7 sind die Zustände $\mathcal{Z}_A$, $\mathcal{Z}_B$, $\mathcal{Z}_C$ und $\mathcal{Z}_D$ homogen, nimmt man die Verwendung einer globalen, zeitbasierten Fenstergröße an. Die Anzahl der Zustandspaare entspricht dann der Anzahl der Eingangsdatenströme ($N_z = N_e = 4$). Der Zustand $\mathcal{Z}_{AB}$, der durch die Zustandspaarsuche ebenfalls als potentieller Zustand für den Zustandstransfer identifiziert wurde, zählt nicht zu dieser homogenen Menge, da er zum einen abhängig von anderen Zuständen ist und sich außerdem dessen Fenstergröße[7] von der Fenstergröße der anderen Zustände unterscheidet und die Gesamtheit der Zustände für den Zustandstransfer deshalb heterogen wäre.

Für die Erläuterung werden erneut die Fälle (a) und (b) als Beispiele genutzt jedoch in Bezug auf mehrere Operatoren. Ferner werden die Varianten (a) und (b) des Systemverhaltens berücksichtigt. Eine Auswahl der resultierenden Kombinationen wird nachfolgend diskutiert.

Als erstes wird Fall (a) in einem System betrachtet, dessen Gesamtverhalten der Variante (a) entspricht. Das bedeutet, dass alle Eingangsdatenströme ihre Daten zeitversetzt um T_v senden und dass Ergebnisse mit jedem Eingangswert errechnet und ausgegeben werden. Als Ausgangsgrößen stehen neben den Systemparametern T_p und T_t auch T_p^* und T_{syst} zur Verfügung, deren Bestimmung am Anfang des Kapitels (Abbildung 5.1, S. 75) beschrieben wurde. Bei schwankungsbeschränkten Datenströmen sind zudem die Schwankungen τ, τ' und der Mindestabstand D zu berücksichtigen. Mit Hilfe dieser Parameter wird T_i errechnet. Für Variante (a) ist Gleichung (5.4a) bei periodischen Eingangsdatenströmen zu benutzen und Gleichung (5.9a) bei schwankungsbeschränkten Eingangsdatenströmen. Schließlich kann mit T_i, T_p und T_t entsprechend Gleichung (5.10) die Anzahl der übertragbaren Datenwerte während einer Pause (N_t) ermittelt werden.

Als nächstes wird die Anzahl der notwendigen Zyklen N_c auf Basis der Gleichung (5.12) berechnet. Die Gleichung ist hinsichtlich der Anzahl der Zustandspaare (N_z) anzupassen. Für Variante (a) ergibt sich N_c wie folgt.

$$N_c = \left\lceil \frac{w_n * N_z - N_i}{N_t + N_i} \right\rceil \tag{5.24}$$

Der Anteil neuer und alter Werte ist anschließend für jeden Zustand zu berechnen. Für die Berechnung der verwendeten neuen Werte jedes Zustands sind zwei Aspekte zu betrachten. Erstens, mit den Eingangswerten welchen Datenstroms wird die Migration abgeschlossen? Zweitens, welcher Eingangswert dieses Datenstroms schließt die Migration ab? Für die Identifikation des Datenstroms ist neben N_c die Anzahl der Eingangs-

[7] Die Anzahl der Werte im Fenster nach Umrechnung von der globalen, zeitbasierten Fenstergröße.

datenströme (N_e) und deren Reihenfolge zu beachten. Die Reihenfolge wird mit dem Parameter d quantifiziert, welcher die Distanz zum Datenstrom mit den ersten Eingangsdaten während der Migration beschreibt. Hat man beispielsweise eine Anfrage mit vier Eingangsdatenströmen in der wiederkehrenden, zeitlichen Reihenfolge $\mathcal{S}_A$, $\mathcal{S}_B$, $\mathcal{S}_C$, $\mathcal{S}_D$ und die Migration beginnt mit Eingangswerten von $\mathcal{S}_C$ dann ist die Distanz für diesen Datenstrom $d = 0$. Der nachfolgende Datenstrom $\mathcal{S}_D$ erhält $d = 1$, $\mathcal{S}_A$ erhält $d = 2$ und $\mathcal{S}_B$ erhält $d = 3$. Für die Migration wird die Vergleichsgröße d' errechnet, mit der man den letzten Eingangsdatenstrom der Migration identifizieren kann.

$$d' = (N_c \mod N_e) \tag{5.25}$$

Für die Berechnung des letzten Eingangswertes wird, wie bei einer Migration mit einem Zustand, die Hilfsgröße N_r genutzt. Die Gleichung (5.13) ist mit der Anzahl der Zustände zu erweitern.

$$N_r = ((w_n * N_z) \mod (N_i + N_t)) \tag{5.26}$$

Schließlich kann mit Hilfe der beiden Vergleichsgrößen N_r und d' für jeden Zustand der Zustandspaarmenge die Anzahl der verwendeten neuen Werte ermittelt werden.

$$N_n = N_i * \left\lfloor \frac{N_c}{N_e} \right\rfloor + \begin{cases} 0, & \text{falls } d > d' \\ N_i, & \text{falls } d < d' \\ N_r, & \text{falls } d = d' \text{ und } 0 < N_r \leq N_i \\ 1, & \text{sonst} \end{cases} \tag{5.27}$$

Die Anzahl der verwendeten alten Zustandswerte errechnet sich wie in Gleichung (5.16) festgelegt ($N_o = w_n - N_n$). Die Berechnung ist für jeden Zustand der Zustandspaarmenge durchzuführen. Für die weitere Verarbeitung werden die relevanten Werte in der Zustandspaartabelle gespeichert. Ein Beispiel einer erweiterten Zustandspaartabelle folgt weiter unten für Fall (b).

Als nächstes wird Fall (b) in einem System mit einem Gesamtverhalten nach Variante (a) betrachtet. In dieser Konfiguration wird ein Ergebnis errechnet, wenn alle N_i Eingangswerte eines Datenstroms empfangen wurden, sodass N_n und N_o immer Vielfache von N_i sind. Die einzelnen Datenströme senden ihre Daten mit einem Abstand von T_v.

Für die Berechnung von N_c wird Gleichung (5.24) verwendet. Die Berechnung von N_r ist nicht nötig. Die Vergleichsgröße d' wird wie in Gleichung (5.25) angegeben ermittelt. Die Berechnung von N_n ist somit eine Vereinfachung der Gleichung für Fall (a) und es bleibt für Fall (b)

$$N_n = N_i * \left\lfloor \frac{N_c}{N_e} \right\rfloor + \begin{cases} 0, & \text{falls } d > d' \\ N_i, & \text{sonst.} \end{cases} \tag{5.28}$$

Die Zustandspaartabelle wird mit den entsprechenden Werten ergänzt. Ein Beispiel ist in Tabelle 5.3 dargestellt. Die Tabelle ist eine Erweiterung des Beispiels aus Tabelle 5.1

(S. 81). Die Werte für d und N_o beziehen sich auf ein Migrationsbeispiel mit der globalen Fenstergröße $w_n^* = 4 * T$ (entspricht $w_n = 12$ bei den Zuständen der Zustandspaarmenge) und einem Migrationsbeginn bei Eingangswerten von $\mathcal{S}_C$. Die erweiterte Zustandspaartabelle ist eine Grundlage für die Auswahl und Konfiguration der jeweiligen Zustandstransferstrategie und wird während der Durchführung der Migration als Hilfsmittel genutzt.

Zustandssender	Zustandsempfänger	d	N_o
$Op_{(AB)}.\mathcal{Z}_A$	$Op_{(A(B(CD)))}.\mathcal{Z}_A$	2	6
$Op_{(AB)}.\mathcal{Z}_B$	$Op_{(B(CD))}.\mathcal{Z}_B$	3	9
$Op_{((AB)C)}.\mathcal{Z}_C$	$Op_{(CD)}.\mathcal{Z}_C$	0	6
$Op_{(((AB)C)D)}.\mathcal{Z}_D$	$Op_{(CD)}.\mathcal{Z}_D$	1	6

Tabelle 5.3: Ergänzung der Zustandspaartabelle mit den Migrationsparametern d und N_o für ein Szenario mit $w_n = 12$, $N_i = 3$, $N_t = 5$ und einem Migrationsbeginn bei Eingangswerten von $\mathcal{S}_C$ (Fall (b), Variante (a))

Als letztes wird die Berechnung der Zustandstransfergrößen in einem System mit einem Gesamtverhalten entsprechend der Variante (b) beschrieben. Bei dieser Variante werden die Werte der Eingangsdatenströme direkt nacheinander in der Anfrage verarbeitet und die Pausendauer T_i ist entsprechend größer im Vergleich zu Variante (a). Ihre Berechnung wurde in den Gleichungen (5.4b) und (5.9b) für periodische bzw. schwankungsbeschränkte Datenströme vorgestellt. Mit T_i wird wiederum N_t gemäß Gleichung (5.10) errechnet. Somit stehen alle Größen zur Berechnung von N_c zur Verfügung. Die Anzahl der notwendigen Zyklen für die Migration ist

$$N_c = \left\lceil \frac{w_n * N_z - N_i * N_z}{N_t + N_i * N_z} \right\rceil . \tag{5.29}$$

Für die weitere Betrachtung dient ein vereinfachtes Szenario mit $N_i = 1$. Eine Konfiguration mit $N_i > 1$ in Kombination mit Variante (b) ist eher ungewöhnlich, weshalb auf eine detaillierte Betrachtung verzichtet wird. Eine rechnerische Lösung für Fälle mit $N_i > 1$ ist jedoch möglich. Für die Berechnung der neuen Werte pro Zustand ist wieder die zeitliche Reihenfolge der Eingangsdatenströme in Form der Distanz d zu beziffern. Zudem wird für die Berechnung der Vergleichsgröße N_r die Gleichung (5.26) an die Eigenschaften der Variante (b) angepasst.

$$N_r = ((w_n * N_z) \mod (N_z + N_t)) \tag{5.30}$$

Damit kann für jeden Zustand der Anteil der neuen Werte und folglich auch der alten Werte (Gleichung (5.16)) berechnet werden, um die Zustandstabelle entsprechend zu ergänzen.

$$N_n = N_c + \begin{cases} 0, & \text{falls } d = 0 \text{ oder } d < N_r \leq N_z \\ 1, & \text{sonst} \end{cases} \tag{5.31}$$

Die Anwendung des Berechnungsmodells auf Anfragen mit mehreren homogenen Zuständen zeigt, dass eine genaue Vorhersage zwar berechenbar ist, aber für jeden Fall individuell unter Berücksichtigung der jeweiligen Konfiguration angepasst werden muss. Hinzu kommt die Erschwernis, dass die Konfiguration vom Startzeitpunkt der Migration abhängt, d. h. mit den Eingangswerten welches Datenstroms begonnen wird. Deshalb kann die Konfiguration erst zur Laufzeit bestimmt werden oder alle möglichen Situationen werden zuvor berechnet, um dann die entsprechende Konfiguration auszuwählen. Für das Migrationsbeispiel wären somit vier Konfigurationen nötig, und zwar für den Migrationsbeginn bei $\mathcal{S}_A$, $\mathcal{S}_B$, $\mathcal{S}_C$ oder $\mathcal{S}_D$.

Heterogene Zustände

Heterogene Zustände unterscheiden sich in mindestens einer Eigenschaft. Für die Nutzung des Berechnungsmodells muss das Systemverhalten trotzdem der Variante (a) oder (b) entsprechen. Damit können die Eigenschaften für die Betrachtung ausgeschlossen werden, die einen Einfluss auf das Systemverhalten haben. Beispielsweise können unterschiedliche Verarbeitungszeiten (T_p) zu einem aperiodischen Systemverhalten führen, weshalb eine Heterogenität der Verarbeitungszeiten hier nicht betrachtet wird.

Eigenschaften, die sich nicht auf das Systemverhalten auswirken und daher für die Migration basierend auf dem Berechnungsmodell in Frage kommen, sind z. B. die Tupeltransferzeit (T_t) und die Fenstergröße, in Form der Anzahl der gespeicherten Werte. Bei anzahlbasierten Fenstern kann ein Unterschied bei der Anzahl der gespeicherten Werte nur bei unterschiedlichen Fenstergrößen auftreten. Bei zeitbasierten Fenstern kann es mehrere Gründe dafür geben, da die Anzahl der gespeicherten Werte nicht nur von der Fenstergröße sondern auch von der Anzahl der Eingangswerte abhängt, die z. B. von der Position des Operators in der Anfrage beeinflusst wird. Dies führt auch bei der Verwendung einer globalen, zeitbasierten Fenstergröße dazu, dass zeitbasierte Fenster auf verschiedenen Ebenen unterschiedliche Anzahlen von Werten speichern. Eine mögliche Umrechnung von einer zeitbasierten Fenstergröße in die entsprechende Anzahl wurde in Gleichung (5.11) (S. 86) gezeigt.

Die nachfolgenden Beispiele sollen einige der zu erwartenden Schwierigkeiten verdeutlichen, die bei der Berechnung der Migrationsparameter für Anfragen mit heterogenen Zuständen auftreten. Dies gilt für abhängige und unabhängige Zustände gleichermaßen.

Bereits bei der Betrachtung von unabhängigen, heterogenen Fenstern kann man beobachten, dass nicht nur die Anzahl der gespeicherten Werte in Betracht gezogen werden muss, sondern auch die zeitliche Größe des Fensters. Man stelle sich eine Anfrage mit zwei gleitenden Fenstern vor. Ein Fenster erhält alle zehn Sekunden fünf Eingangswerte und speichert diese in einem zeitbasierten Fenster mit einer Größe von fünf Minuten ($N_i = 5$, $T = 10\,\text{s}$, $w_1^* = 5\,\text{min}$). Die Anzahl der darin gespeicherten Werte beträgt somit

150. Das andere Fenster erfasst alle zehn Sekunden zehn Eingangswerte über einen Zeitraum von einer Stunde ($N_i = 10$, $T = 10\,\mathrm{s}$, $w_2^* = 60\,\mathrm{min}$). Somit werden 3600 Werte in diesem Fenster gespeichert. Wird nun eine Migration mit $N_t = 50$ durchgeführt, dauert diese allein für das zweite Fenster 10 Minuten. Da die Migration länger dauert, als das erste Fenster groß ist, ist für dieses Fenster kein Zustandstransfer nötig. Es kann während der Migration ausschließlich mit neuen Werten gefüllt werden. Die vorgestellten Berechnungsformeln sind für ein solches Szenario nicht geeignet.

Das gleiche gilt für die Berücksichtigung von abhängigen Zuständen bei der Migration. Ein Beispiel findet sich in Abbildung 5.7. Der Zustand $\mathcal{Z}_{AB}$ erhält für jeden Eingangswert von $\mathcal{S}_A$ und $\mathcal{S}_B$ mehrere Eingangswerte abhängig von den Fenstergrößen und vom Algorithmus im vorherigen Operator ($A \bowtie B$). Deshalb ist ein Zustandstransfer für $\mathcal{Z}_{AB}$ sinnvoll, wenn sich sein Zustand durch Transfer schneller füllen lässt als durch Berechnung, d. h. wenn die Verarbeitungszeit im vorherigen Operator die Transferzeit für die entsprechende Anzahl von Ergebnis- bzw. Eingangstupeln übersteigt. Die Schwierigkeit ist, dass N_o für $\mathcal{Z}_{AB}$ einerseits von N_c abhängt und andererseits N_c beeinflusst. Je mehr Zyklen für die Migration benötigt werden, desto mehr Eingangswerte von $\mathcal{S}_A$ und $\mathcal{S}_B$ werden verarbeitet. Dies bedeutet ebenso mehr Eingangswerte für $\mathcal{Z}_{AB}$. Dadurch sind weniger Werte aus dem Zustandstransfer nötig, was gleichzeitig weniger Zyklen für die Migration zur Folge hat.

Eine weitere Schwierigkeit zeigt sich, wenn die System- und Zustandseigenschaften so sind, dass sich für jeden Zustand eine individuelle Tupeltransferzeit ergibt, beispielsweise durch verschiedene Tupelgrößen oder Netzwerkverbindungen. Durch unterschiedliche T_t ist N_t nicht mehr konstant, weil in jeder Pause eine unterschiedliche Zusammensetzung von Tupeln übertragen werden kann. Das vorgestellte Berechnungsmodell ist für diese Anforderung nicht ausgelegt. Zudem bedeutet eine Optimierung, mit dem Ziel, alle Pausen bestmöglich auszunutzen und die Migrationszeit zu minimieren, einen erhöhten Berechnungsaufwand. Das Finden einer entsprechenden Optimierung ist nicht Gegenstand dieser Arbeit.

Alle drei Beispiele zeigen, dass das numerische Verfahren zur Migrationsvorhersage bei heterogenen Zuständen an seine Grenzen stößt. Bereits die Variation von einzelnen Eigenschaften würde sehr spezifische Anpassungen des Berechnungsmodells erfordern. Ein allgemeines Modell scheint mit angemessenem Aufwand nicht realisierbar, da zu viele Eigenschaften berücksichtigt werden müssen. Ein heuristischer Lösungsansatz wird im nächsten Kapitel vorgestellt.

5.2.3 Zusammenfassung Zustandsauswahl

In diesem Unterabschnitt wurde die allgemeine Vorgehensweise bei der Berechnung der Zustandstransferparameter innerhalb periodischer oder schwankungsbeschränkter periodischer Datenströme vorgestellt. Für Einzelzustände und für Anfragen mit mehreren homogenen Zuständen wurde mit zahlreichen Beispielen gezeigt, dass eine Berechnung prinzipiell möglich ist, die Vielfältigkeit der Eigenschaften von Anfragen und des Sys-

tems jedoch Variationen der Berechnungsformeln erfordern. Mit zunehmender Anzahl von Zuständen und bei Berücksichtigung unterschiedlicher Eigenschaften wird das Berechnungsmodell komplizierter. Insbesondere heterogene Zustände und die Abhängigkeit von Zuständen stellen große Herausforderungen bei der Berechnung der Migrationsparameter dar.

Allerdings ist die generelle Vorgehensweise bei allen vorgestellten Beispielen identisch. Über die Berechnung der notwendigen Migrationszyklen (N_c) kann der Anteil von neuen und alten Tupeln (N_n und N_o) innerhalb eines Fensters bei Migrationsende errechnet werden. Optional kann eine Anzahl von Zustandswerten am Anfang der Migration (N_a) übertragen werden. Gegebenenfalls sind für die Berechnung der Migrationsparameter Hilfsgrößen wie d', N_r, N_a^* und N_a' zu benutzen. Die Tabelle 5.4 enthält eine Auflistung aller Größen, die aus den Ausgangsgrößen (Tabelle 5.2) abgeleitet wurden oder sich während der Identifikation der Transfermenge ergeben.

Formelzeichen	Beschreibung
T_i	Pausendauer
N_t	Anzahl übertragbarer Werte während einer Pause
N_c	Anzahl notwendiger Zyklen für die Migration
N_n	Anzahl aller neuen Werte im Zustand bei Migrationsende
N_o	Anzahl aller übertragenen Werte im Zustand bei Migrationsende
N_a	Anzahl transferierbarer Zustandswerte am Anfang der Migration
d	Distanz von Eingangsdatenströmen (Reihenfolge)
d'	Hilfsgröße zur Berechnung des letzten Eingangsstroms der Migration
N_r	Hilfsgröße zur Berechnung des letzten Eingangswertes der Migration
N_a'	Vergleichsgröße bei sofortigem Zustandstransfer
N_a^*	Hilfsgröße bei sofortigem Zustandstransfer

Tabelle 5.4: Migrationsparameter und Hilfsgrößen

Als Ergebnis der Berechnungen wird die Zustandspaartabelle auf die notwendigen Zustände reduziert und um zusätzliche Informationen erweitert. Hervorzuheben sind N_o und d, die später als Ausgangsgrößen für die Festlegung des lokalen und globalen Transferverhaltens genutzt werden. Wenngleich das vorgestellte Modell für die Berechnung von Migrationsparametern in verteilten Umgebungen gedacht und ausgelegt ist, kann es ebenso für einen lokalen Zustandstransfer angewendet werden, indem $T_t = 0$ gesetzt wird oder entsprechend kleine Werte benutzt werden.

5.3 Zustandstransferstrategien

Mit Kenntnis der zu übertragenden Zustände und der jeweiligen Anzahl der zu übertragenden Werte (N_o) kann für jeden Zustand das lokale Transferverhalten und für die Menge der Zustände das globale Transferverhalten festgelegt werden. Das lokale Transferverhalten wird durch die Zustandstransferstrategien definiert und bestimmt jeweils die Reihenfolge, in der Zustandswerte eines Zustandes übertragen werden. Das globale Transferverhalten bestimmt die Reihenfolge, in der die Transfers der einzelnen Zustände ausgeführt werden sowie die Anzahl der Werte, die pro Aufruf übertragen werden.

5.3.1 Lokales Transferverhalten

Mit N_o wurde für jeden Zustand die Anzahl von Transferwerten errechnet, welche die schnellstmögliche Migration gewährleistet. Die tatsächlich zur Verfügung stehenden Werte wurden dabei zunächst nicht berücksichtigt. Dies wird nun mit der Auswahl einer Zustandstransferstrategie getan, um schließlich die exakte Anzahl von Transferwerten zu berechnen. Dabei sind die Eigenschaften und die Dynamik des Fensters des Zustandssenders zu beachten. Es sei erinnert, dass verworfene Zustandswerte nicht übertragen werden können. Sind weniger als N_o Werte verfügbar, müssen mehr neue Werte, als zuvor berechnet, benutzt werden, um das Fenster zu vervollständigen und die Migration abzuschließen.

Damit eine Zustandstransferstrategie die Transferwerte aus der korrekten Untermenge auswählt, ist eine Sortierung der im Fenster enthaltenen Werte nötig. Es ist die Reihenfolge herzustellen, in der die Datentupel aus dem Zustand verworfen werden. Im Fall von gleitenden Fenstern ist somit die zeitliche Reihenfolge interessant. Bei einfachen Datentupeln, z. B. von Eingangsdatenströmen, wird der Zeitstempel der Datentupel als Sortierkriterium genutzt. Sofern die Eingangsdaten in der Reihenfolge ihres Auftretens verarbeitet werden (FIFO), sind derartige Zustände bereits sortiert. Bei zusammengesetzten Datentupeln ist der minimale Zeitstempel für die Sortierung relevant. Wird ein anderes Kriterium zum Verwerfen der Datentupel genutzt, sind die Werte des Zustands diesem Kriterium entsprechend zu sortieren.

Da eine direkte Sortierung des Zustandes nicht praktikabel ist, muss ein Abbild des Zustandes geschaffen werden, welches sortiert und für den Zustandstransfer genutzt werden kann. Dies kann beispielsweise durch die Erzeugung eines Index bewerkstelligt werden. Die Dynamik des Fensters, z. B. durch das Aktualisieren des Fensters bei eintreffenden neuen Werten oder beim Verwerfen von Werten, ist bei der Verwaltung des Index zur Laufzeit zu beachten. Die neuen Werte müssen jedoch nicht zum Index hinzugefügt werden, da der Zustandsempfänger diese Werte aus den Eingangsdatenströmen erhält. Eine andere Möglichkeit ist das Kopieren des Zustands in eine neue Datenstruktur. Die Datentupel können dann separat sortiert werden und sind unabhängig von der Dynamik des Fensters. Überflüssige Werte der Zustandskopie können sofort verworfen werden. Im Vergleich zu einem Index benötigt diese Möglichkeit mehr Speicher und wird im

Folgenden nicht betrachtet.

Eine Zustandstransferstrategie wird auf die sortierte Menge von Zustandswerten angewendet. Das älteste Datentupel der sortierten Menge ist jenes, welches als nächstes verworfen wird. Das neueste Datentupel dieser Menge würde als letztes verworfen. Relevant für den Transfer sind die N_o neuesten Werte – sofern verfügbar.

Für jede Zustandstransferstrategie wird die Größe $w_{o,*}$ ermittelt oder abgeschätzt. Diese beschreibt, wie groß das Fenster des Zustandssenders (w_o) mindestens sein muss, um N_o Werte zu übertragen. Erfüllt das Fenster im Zustandssender die Mindestgröße ($w_o \geq w_{o,*}$), dann wird die Migration in der kürzestmöglichen Zeit durchgeführt. Ist es kleiner, sind zusätzliche neue Werte notwendig, um das Fenster des Zustandsempfängers zu vervollständigen und um die Migration abzuschließen.

Nachfolgend werden verschiedene Zustandstransferstrategien beschrieben. Die dabei benutzten Bezeichner entsprechen den Abkürzungen der englischen Strategienamen, wie sie bereits publiziert wurden [WB10, WHB+10], weshalb auf deutsche Namen und Kurzformen verzichtet wird.

LF – Latest First Mit der Zustandstransferstrategie „Latest First“ (LF) werden die Werte aus der sortierten Menge beginnend mit dem neuesten Wert übertragen (LIFO). Der Transfer wird solange mit älteren Datentupeln fortgesetzt, bis entweder N_o Werte übertragen wurden oder keine Werte mehr zur Verfügung stehen, da sie bereits verworfen wurden. Die zweite Bedingung stellt gleichzeitig das Problem von LF dar. Durch die Übertragungsreihenfolge können alte Datentupel nicht vor dem Verwerfen bewahrt werden. Ein beispielhaftes Schema befindet sich in Abbildung 5.8 (LF). Für jede der abgebildeten Zustandstransferstrategien ist die Zusammensetzung des neuen Zustandes aus Eingangswerten („neu“) und Zustandstransferwerten („alt“) nach Abschluss des Zustandstransfers[8] dargestellt. Ein Eingangswert und die danach übertragenen Transferwerte sind jeweils im gleichen Grauton dargestellt. Die Pfeile der Transferstrategien symbolisieren die Transferrichtung. Offene Plätze sind mit neuen Werten zu füllen, da keine weiteren Transferwerte zur Verfügung stehen.

Für Fenster von Eingangsdatenströmen hat LF die Eigenschaft, dass die neuen Werte und die Transferwerte ein zusammenhängendes Fenster bilden. Für zusammengesetzte Datentupel muss das nicht gelten. Ursache dafür ist die Sortierung der Datentupel nach dem Zeitpunkt des Verwerfens, die nicht der Zeitstempelreihenfolge der Datentupel entsprechen muss. Ein zusammenhängendes Fenster hat den Vorteil, dass man über die entsprechende Zeitspanne eine präzise Aussage treffen kann, z. B. bei der Berechnung eines gleitenden Mittelwertes. Für segmentierte Fenster sind allenfalls Näherungen möglich, da Lücken in der Zeitstempelreihenfolge entstehen, wo Werte erst später transferiert werden. Ein zusammenhängendes Fenster kann nützlich sein, wenn der Wechsel zur neuen Anfrage noch während des Zustandstransfers geplant ist[9] und die Anforderung

[8] nicht notwendigerweise Migrationsende

[9] beispielsweise bei einer Fenstervergrößerung

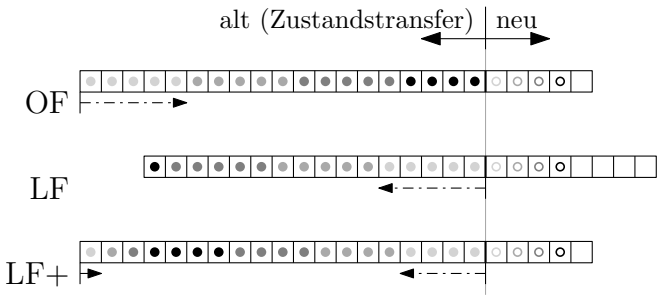


Abbildung 5.8: Vergleich der Zustandstransferstrategien OF, LF und LF+; Fensterinhalt nach Abschluss des Zustandstransfers für $w_n = 24$, $w_o = 20$ und $N_t = 5$ bei Betrachtung der Migration eines Einzelzustandes

besteht, präzise Werte zu berechnen.

Die minimale Fenstergröße, mit der die optimale Migration mittels LF erreicht werden kann, ist $w_{o,lf}$. Ihre Berechnung lautet für die Migration eines einzelnen, gleitenden Fensters

$$w_{o,lf} = N_o + \left\lceil \frac{N_o}{N_t} \right\rceil * N_i \tag{5.32}$$

Bei der Migration von mehreren Zuständen kann $w_{o,lf}$ lediglich abgeschätzt werden. Eine genaue Berechnung ist nur bei Betrachtung des Einzelfalls möglich, da zu viele Eigenschaften diese Größe beeinflussen. Einige Beispiele dafür lassen sich bereits bei Bäumen von Verbundoperatoren beobachten. So hängt beim Zustand eines Verbundoperators die aktuelle Anzahl von Werten nicht vom eigenen Datenstrom ab, sondern vom anderen Eingangsstrom des Operators, weil dieser das Verwerfen auslöst. Daraus folgt, dass sich zeitweilig $w_o + N_i$ Werte in Fenstern der Zustandssender befinden, also mehr Werte, als mit dem Fenster betrachtet werden. Diese Werte stehen ebenfalls für den Zustandstransfer zur Verfügung, sofern sie rechtzeitig übertragen werden können. Dies hängt neben N_t von der Reihenfolge der Eingangsdatenströme und dem Beginn der Migration ab. Beispielsweise gilt für die Originalanfrage in Abbildung 5.7, dass die Zustände $\mathcal{Z}_B$, $\mathcal{Z}_C$ und $\mathcal{Z}_D$ bei Eingangswerten von $\mathcal{S}_A$ bereinigt werden, setzt man eine Reihenfolge der Eingangsdatenströme von $\mathcal{S}_A$, $\mathcal{S}_B$, $\mathcal{S}_C$ und $\mathcal{S}_D$ sowie eine globale Fenstergröße voraus. Der Datenstrom $\mathcal{S}_B$ hat die Bereinigung des Zustands $\mathcal{Z}_A$ zur Folge und Eingangswerte von $\mathcal{S}_C$ und $\mathcal{S}_D$ haben keinen Einfluss auf Zustände der Zustandspaarmenge. Beginnt eine Migration mit Eingangswerten von $\mathcal{S}_C$ werden über zwei Zyklen keine Werte aus Transferzuständen verworfen, wohingegen ein Beginn bei $\mathcal{S}_A$ sofort das Verwerfen von Datentupeln in drei Zuständen nach sich zieht.

Für homogene Zustände von gleitenden Fenstern gilt

$$\max(\{N_o\}) + (N_c^* - 1) * N_i \leq w_{o,lf} \leq w_n \quad \text{für } N_c^* > 0 \tag{5.33}$$

mit

$$N_c^* = \begin{cases} \lfloor N_c/N_e \rfloor, & \text{für Variante (a)} \\ N_c, & \text{für Variante (b).} \end{cases} \tag{5.34}$$

Die Abschätzung beschränkt sich auf die beschriebenen Fälle. Für die untere Schranke wird der größte Wert für N_o aus der Zustandspaartabelle verwendet. Addiert wird die Anzahl der Werte, die während der Migration verworfen werden. Mit $N_c^* - 1$ wird dem Verhalten der Verbundoperatoren Rechnung getragen. Werden ausschließlich Operatoren verwendet, die ihre Fenster direkt beim Eintreffen neuer Werte aktualisieren, kann N_c^* statt $N_c^* - 1$ verwendet werden. Die obere Schranke ist w_n.

OF – Oldest First Bei der Zustandstransferstrategie „Oldest First“ (OF) überträgt der Zustandssender die Datentupel beginnend mit dem ältesten Wert der Transfermenge (FIFO). Abhängig von w_o kann dieser Wert mitten im Fenster liegen, weshalb die Kenntnis von N_o bei OF wesentlich ist. Stehen weniger als N_o Werte im Zustand zur Verfügung, wird beim ältesten verfügbaren Wert begonnen. Der Transfer endet, wenn der letzte Wert der Transfermenge übertragen wurde. Im Unterschied zu LF ist es mit OF möglich, Datentupel vor dem Verwerfen zum Zustandsempfänger zu übertragen.

In Abbildung 5.8 (OF) ist der Zustandstransfer mittels OF dargestellt. Die ältesten Zustandswerte (hellgrau) wurden zuerst übertragen und sind zum dargestellten Zeitpunkt im Zustandssender bereits verworfen worden. Der einzige freie Platz wird mit dem nächsten neuen Wert gefüllt, wodurch die Migration abgeschlossen ist. Verglichen mit LF werden mehr Zustandswerte übertragen, und die Migration kann eher beendet werden.

Durch die Verwendung von OF wird das Fenster beim Zustandsempfänger solange eine Lücke aufweisen, bis der letzte Transferwert übertragen wurde. Falls dies in der Anwendung Probleme bewirkt, z. B. bei der Interpretation von Ergebnissen, ist diese Eigenschaft als Nachteil zu bewerten. Wird erst nach dem vollendeten Zustandstransfer zur neuen Anfrage gewechselt, ist das allerdings kein Problem.

Die minimale Fenstergröße zum Erreichen der schnellstmöglichen Migration mittels OF ist $w_{o,of}$. Für ein einzelnes, gleitendes Fenster beträgt sie

$$w_{o,of} = N_o + N_i. \tag{5.35}$$

Für Anfragen mit homogenen Zuständen lautet die Abschätzung

$$\max(\{N_o\}) - N_i \leq w_{o,of} \leq w_n - N_c^* * N_i. \tag{5.36}$$

N_c^* gilt entsprechend Gleichung (5.34). Auch hier wird die untere Schranke durch das Verhalten der Verbundoperatoren beeinflusst. Sind lediglich Operatoren vorhanden, die direkt verwerfen, dann entspricht die untere Schranke mindestens der größten Anzahl N_o der Zustandspaarmenge. Ob die untere Schranke erreicht werden kann, hängt im Wesentlichen davon ab, wann und wie viele Werte verworfen werden, und ob alle diese

Werte zuvor transferiert werden können. Ist dies nicht realisierbar, ist $w_{o,of}$ entsprechend größer. Die obere Schranke ist von N_c^* abhängig, da mit jedem durchgeführten Zyklus weniger Zustandswerte übertragen werden müssen und demzufolge das Fenster des Zustandssenders kleiner sein kann.

Additional Oldest Value Preservation Diese Methode ist keine eigenständige Strategie, sondern eine Erweiterung für einige Zustandstransferstrategien. Die Aufgabe der Methode ist es, in jedem Zyklus zu Beginn des Zustandstransfers die ältesten Werte der Transfermenge zu übertragen. Dadurch kann der Zustandsempfänger diese Werte erhalten, bevor sie beim Zustandssender verworfen werden. Die Erweiterung wird symbolisiert durch ein +. Wird sie beispielsweise auf LF angewendet, entsteht die Zustandstransferstrategie LF+. Mit LF+ werden anfangs jeweils N_i alte Werte der Transfermenge transferiert. In der verbleibenden Zeit während der Pause werden Zustandswerte aus der Transfermenge mittels LF übertragen. Ein Beispiel für $N_i = 1$ ist in Abbildung 5.8 (LF+) dargestellt. Mit LF+ können die Vorteile von OF und LF vereint werden. Man ist mit LF+ in der Lage, eine gleichgroße Transfermenge wie mit OF zu übertragen. Lediglich die Übertragungsreihenfolge der Transferwerte unterscheidet sich. Es können Zustandswerte vor dem Verwerfen transferiert werden, wodurch die Migration schneller endet als mit LF. Zudem ist die Menge der zusammenhängenden neuen Werte größer als bei einem Transfer durch OF. Es gelten für $w_{o,lf+}$ die Gleichungen (5.35) und (5.36) für Einzelzustände bzw. für homogene Zustände.

RS – Random Selection Mit der Strategie „Random Selection“ (RS) werden Werte zufällig aus der Transfermenge ausgewählt und übertragen. Dabei ist jede diskrete Verteilungsfunktion als Basis für die Auswahl denkbar. Der Vorteil der Strategie ist, dass mit geringem Aufwand eine Verteilung von Werten über das gesamte Fenster erreicht werden kann. Sofern Ergebnisse der neuen Anfrage bereits während der Migration relevant sind und genäherte Ergebnisse akzeptabel sind, kann RS dazu genutzt werden, bei Verwendung einer geeigneten Verteilungsfunktion eine Abschätzung über das gesamte Fenster zu erhalten, z. B. bei der Berechnung eines gleitenden Mittelwertes. Im Vergleich dazu können Ergebnisse mit LF oder OF verschoben sein, da diese Strategien Sequenzen von neuen bzw. alten Werten übertragen und somit anfällig für Ergebnisabweichungen sind, beispielsweise wenn wachsende oder fallende Signale analysiert werden.

RS hat den Nachteil, dass Datenwerte mehrfach übertragen werden können, da die Auswahl ohne Rücksicht auf die bereits übertragenen Werte erfolgt. Außerdem kann nicht garantiert werden, dass alle Werte der Transfermenge übertragen werden. Wie bei LF können Werte verworfen werden, bevor sie für die Übertragung ausgewählt wurden. Diese Eigenschaft kann dazu führen, dass die Migrationsdauer im schlechtesten Fall der Anlaufzeit des Operators entspricht, zumindest dann, wenn die Umschaltbedingung ein lückenlos gefülltes Fenster beim Zustandsempfänger ist.

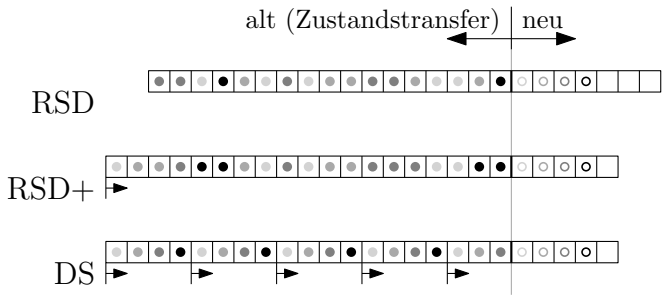


Abbildung 5.9: Vergleich der Zustandstransferstrategien RSD, RSD+ und DS; Fensterinhalt nach Abschluss des Zustandstransfers für $w_n = 24$, $w_o = 20$ und $N_t = 5$ bei Betrachtung der Migration eines Einzelzustandes

RSD – Random Selection without Doubles Eine Erweiterung von RS ist die Strategie „Random Selection without Doubles“ (RSD). Mit RSD wird eine Mehrfachübertragung von Werten verhindert. Damit wird ein großer Nachteil von RS kompensiert. Trotzdem kann die Übertragung aller Werte nicht garantiert werden. Durch die zufällige Auswahl kann es geschehen, dass Werte verworfen werden, ohne dass sie zuvor ausgewählt und zum Zustandsempfänger übertragen wurden. Die Wahrscheinlichkeit dafür, ist bei den ältesten Werten am größten.

Für die Abschätzung von $w_{o,rsd}$ sind die zwei Extremfälle zu betrachten. Im günstigsten Fall können alle Tupel der Transfermenge übertragen werden, so wie bei OF. Der ungünstigste Fall ist, wenn keines der Tupel, die während der Migration verworfen werden, zuvor zum Zustandsempfänger übertragen werden kann. Die Menge der transferierten Tupel würde dann der Transfermenge von LF entsprechen. Deshalb ist $w_{o,rsd}$ die Vereinigung der Intervalle für OF und LF. Das bedeutet für einen Einzelzustand

$$N_o + N_i \leq w_{o,rsd} \leq N_o + \left\lceil \frac{N_o}{N_t} \right\rceil * N_i \tag{5.37}$$

und für homogene Zustände

$$\max(\{N_o\}) - N_i \leq w_{o,rsd} \leq w_n. \tag{5.38}$$

Ein Beispiel für RSD befindet sich in Abbildung 5.9. Die zwei ältesten Werte wurden nicht rechtzeitig ausgewählt, um übertragen zu werden. Deshalb muss das Fenster des Zustandsempfängers mit zwei zusätzlichen neuen Werten gefüllt werden, wodurch die Migration länger dauert.

Durch die Erweiterung von RSD mit der Erhaltungsmethode für die ältesten Werte kann dieser Effekt vermieden werden. Die resultierende Zustandstransferstrategie ist RSD+. Der mittlere Fall in Abbildung 5.9 enthält ein entsprechendes Beispiel. RSD+ besitzt die gleichen Transfereigenschaften wie OF oder LF+, weshalb eine schnellstmögliche Migration mit RSD+ realisierbar ist. Die Größe $w_{o,rsd+}$ kann deshalb entsprechend

den Gleichungen (5.35) und (5.36) für Einzelzustände bzw. für homogene Zustände bestimmt werden.

RSD und RSD+ haben wie RS den Vorteil, eine Verteilung über das gesamte Fenster zu ermöglichen und dadurch für manche Operationen während der Migration eine bessere Näherung als OF oder LF zu erlauben.

DS – Distributed Selection Die Zustandstransferstrategie „Distributed Selection“ (DS) erlaubt eine systematische Auswahl von Werten aus der Transfermenge. So ist es möglich, Werte mit einem angegebenen Abstand zu übertragen, z. B. jeder vierte Wert. In jeder Pause wird dann eine andere Sequenz von Werten übertragen, bis sich alle Werte der Transfermenge im Zustand des Empfängers befinden. In Abbildung 5.9 (DS) ist ein entsprechendes Beispiel abgebildet.

Für Fälle mit $N_i = 1$ und bei korrekter Konfiguration des Abstandes, kann bereits mit DS die schnellstmögliche Migration erreicht werden. Der Abstand zwischen den zu übertragenden Werten (d_{DS}) muss dabei mindestens so groß gewählt werden, dass sich die Menge der Transferwerte einer Pause über die gesamte Transfermenge erstreckt, d. h. $d_{DS} \geq \lceil N_o/N_t \rceil$. Zudem muss die Position des letzten Wertes einer Pause gespeichert werden, um als Ausgangswert für die Übertragung der nächsten Pause genutzt werden zu können.

Um den Verlust von alten Werten speziell bei $N_i > 1$ zu vermeiden, sollte die erweiterte Strategie DS+ genutzt werden. Der Abstand ist dann entsprechend der verbleibenden Transferkapazität zu konfigurieren. Die Größe $w_{o,ds+}$ ist gemäß den Gleichungen (5.35) und (5.36) für Einzelzustände bzw. für homogene Zustände bestimmbar.

Prinzipiell ist es auch denkbar, nicht-homogene Verteilungen für die Auswahl zu verwenden, d. h. die Abstände der ausgewählten Werte sind dann nicht gleich groß. In diesem Fall sollte DS+ für die Übertragung eingesetzt werden, um die ältesten Werte zu erhalten.

SemS – Semantic Selection Die Transferstrategie „Semantic Selection“ (SemS) unterscheidet sich von den bisher beschriebenen Strategien. Diese Strategien sind im Unterschied zu SemS mit einfachen Mechanismen realisierbar. Die dafür notwendigen funktionalen Erweiterungen werden im nächsten Abschnitt (5.4) beschrieben. Hinzu kommt unter Umständen eine Sortierung beim Einfügen in den neuen Zustand. Weitere Kenntnisse sind für den Zustandsempfänger nicht notwendig. Für SemS benötigt der Empfänger zusätzliche Informationen und Anweisungen. Die Semantik der empfangenen Daten muss ihm bekannt sein.

Zwei Beispiele sollen das Potential von SemS verdeutlichen. Im ersten Beispiel wird die Migration eines Operators zur Berechnung eines gleitenden Mittelwertes betrachtet. Für die Berechnung eines gleitenden Mittelwertes gibt es mehrere Möglichkeiten. Eine einfache aber wenig effiziente Methode ist, in jedem Zyklus die Werte des Fensters zu summieren und den Mittelwert zu berechnen. Effizienter ist die Berechnung basierend auf

einer gleitenden Summe des Fensters, von der verworfene Werte abgezogen werden und neue hinzu addiert. Dadurch muss das Fenster niemals vollständig ausgewertet werden. Im Beispielszenario ist der einfache Operator Teil der Originalanfrage und wird mittels Migration durch einen effizienten Operator ersetzt. Durch die Verwendung von SemS ist es möglich, die Migration in äußerst kurzer Zeit abzuschließen. Für die Inbetriebnahme des neuen Operators sind lediglich drei Werte notwendig. Der oder die nächsten Eingangswerte, die der neue Operator vom Eingangsdatenstrom erhält. Zudem benötigt er von Zustandssender die aktuelle Summe des Fensters und den oder die Werte, die als nächstes verworfen werden. Damit kann der neue Operator korrekte Ergebnisse erzeugen und umgehend in Betrieb genommen werden. Für den weiteren Verlauf muss garantiert sein, dass alle Werte die als nächstes verworfen werden rechtzeitig an den neuen Operator übertragen werden, d. h. der Zustandstransfer wird nach der Inbetriebnahme des neuen Operators fortgesetzt bis alle Werte übertragen wurden.

Bei der Betrachtung von Zustandssender und Zustandsempfänger sind die unterschiedlichen Aufgaben und das Zusammenwirken beider Operatoren zu erkennen. Der Zustandssender muss zunächst die Summe über alle Werte seines Zustandes berechnen und zum Zustandsempfänger übertragen. Damit dieser den Wert von anderen Datenwerten unterscheiden kann, muss wenigstens dieser Wert besonders gekennzeichnet sein. Dem Zustandsempfänger muss die Semantik bekannt sein, damit er den Wert entsprechend verarbeiten kann, d. h. in seinem Speicher für die gleitende Summe ablegen. Danach kann der Transfer der Zustandswerte durchgeführt werden, wofür eine der beschriebenen Transferstrategien verwendet werden kann, z. B. OF.

Das zweite Beispiel bezieht sich auf die Migration eines Operators, der den Maximalwert eines gleitenden Fensters bestimmt. Für den Zustand des neuen Operators sind lediglich die Maximalwerte relevant. Alle anderen Werte werden von diesen Werten dominiert und müssen deshalb nicht übertragen werden, da sie nie zu einem Ergebnis beitragen. Die Transfermenge kann deshalb auf die dominierenden Werte reduziert werden. Der Zustandssender hat diese Aufgabe zu übernehmen und die Werte in einer geeigneten Reihenfolge zu übertragen. Der Empfänger hat die Werte entsprechend in seinem Zustand zu speichern, insbesondere, wenn es sich um ein anzahlbasiertes Fenster handelt. Auch hier wird deutlich, dass Zustandssender und Zustandsempfänger über Informationen und Methoden verfügen müssen, die den Funktionsumfang der einfachen Zustandstransferstrategien überschreiten.

Mit SemS besteht die Möglichkeit, dass ein Wechsel zu einer neuen Anfrage innerhalb eines Zyklus realisiert werden kann. Der Zustandstransfer kann dann im Hintergrund fortgeführt werden. Ebenso gibt es Anwendungsfälle, die die Reduktion der Transfermenge auf wenige notwendige Werte erlauben. Ein grundlegender Vorteil von SemS ist, dass sich die Migration an Anforderungen anpassen lässt, die mit den einfachen Zustandstransferstrategien nicht lösbar wären. Der Nutzen ist allerdings auf besondere Fälle, wie die oben beschriebenen Beispiele, beschränkt. Im Vergleich zu den einfachen Strategien ist eine funktionale Erweiterung des Zustandsempfängers erforderlich, die über die gewöhnlichen Erweiterungen für den Zustandstransfer hinausgehen. SemS ist prinzipiell in

jedem Migrationsszenario anwendbar, jedoch ist in vielen Fällen der Aufwand geringer, greift man auf eine der einfachen Zustandstransferstrategien zurück.

Zusammenfassung Das lokale Transferverhalten wird für jeden Zustand durch die Auswahl einer Zustandstransferstrategie festgelegt. Zur Verfügung stehen sowohl einfache Zustandstransferstrategien (OF, LF, LF+, RS, RSD, RSD+, DS, DS+) als auch die Strategie SemS, die den Gegebenheiten entsprechend mit zusätzlichen Funktionen versehen werden kann.

Wenngleich die Auswahl einer Transferstrategie entsprechend den Benutzeranforderungen erfolgt, sind die Eigenschaften der Strategien zu beachten, um eine optimale Durchführung der Migration zu erreichen. Die Strategie SemS sollte in Betracht gezogen werden, wenn für die zu transferierende Anfrage die Möglichkeit besteht, die Migration durch die Verwendung von SemS maßgeblich zu beschleunigen, z. B. durch schnelleres Umschalten oder durch Reduktion der Transfermenge. Der entstehende Mehraufwand für die Konfiguration von SemS ist zu berücksichtigen. Ist SemS nicht anwendbar, kann eine der anderen Strategien gewählt werden.

Für die Auswahl einer Strategie ist von Bedeutung, unter welchen Bedingungen zur neuen Anfrage gewechselt wird, d. h. ob die neue Anfrage bereits während der Migration Ergebnisse produziert oder ob erst nach dem Zustandstransfer oder am Migrationsende zur neuen Anfrage gewechselt wird. Bei Ergebnisausgabe während der Migration ist zu beurteilen, welche Auswirkung die Auswahl einer Strategie auf die Berechnung des Endergebnisses hat. Wird während der Migration eine Näherung über das gesamte Fenster benötigt, dann bieten sich RS- und DS-basierte Strategien an. Für sequenzielle Übertragungen sind eher OF, LF oder LF+ interessant. Werden keine Ergebnisse während der Migration ausgegeben, entfällt dieser Aspekt als Entscheidungskriterium.

Ein weiterer Gesichtspunkt für die Auswahl einer Transferstrategie ist die Anzahl der zur Verfügung stehenden Werte im Zustand des Zustandssenders, bestimmt durch w_o und evtl. Eigenschaften des Systems und der Anfrage, wie sie beispielsweise bei Verbundbäumen vorkommen. Für Migrationsfälle mit $w_o \geq w_n$ ist immer eine schnellstmögliche Migration möglich, da unabhängig von der Transferstrategie immer hinreichend viele Werte im Zustand des Zustandssenders zur Verfügung stehen. Anwendungsfälle dafür sind Fensterverkleinerungen oder Migration ohne Änderung der Fenstergröße, z. B. Verschiebung von Operatoren auf andere Verarbeitungsknoten.

Falls $w_o < w_n$, dann ist anhand von $w_{o,*}$ zu überprüfen, ob eine optimale Migration möglich ist und welche Strategien dafür besonders geeignet sind. Die Größe $w_{o,*}$ wurde für jede Strategie angegeben und basiert auf N_o, der Anzahl von notwendigen Werten, die in der Zustandspaartabelle gespeichert ist. Bei $w_o \geq w_{o,*}$ ist die Migration schnellstmöglich realisierbar. Andernfalls sind zusätzliche neue Datenwerte aus dem Eingangsdatenstrom notwendig, um die offenen Plätze im Zustand zu vervollständigen. Erst dann kann ein exaktes Ergebnis berechnet werden.

Die Betrachtung von Migrationsszenarien mit mehreren Zuständen hat gezeigt, dass

$w_{o,*}$ für diese nur abgeschätzt werden kann. Ursache ist, dass die tatsächliche Konfiguration erst zur Laufzeit ermittelt oder ausgewählt werden kann, da sie z. B. davon abhängt, mit welchen Eingangswerten die Migration beginnt. Außerdem verdeutlicht die Analyse von Verbundbäumen, dass unter Umständen mehr Datentupel in Zuständen zur Verfügung stehen und für den Transfer genutzt werden können, als die Fenstergröße angibt. Um eine genaue Aussage über die Migration treffen zu können und Konfigurationsparameter abzuleiten, ist der Einzelfall zu betrachten oder ein anderes Verfahren anzuwenden[10].

Für alle Strategien außer LF und SemS ist die Kenntnis von N_o von besonderer Bedeutung, da dadurch die Größe der Transfermenge definiert ist. Stehen weniger Werte im Zustand des Senders zur Verfügung, ist N_o in der Zustandspaartabelle entsprechend zu korrigieren, da dieser Wert als Parameter zur Konfiguration der Migration genutzt wird. Bei LF kann auf die Bestimmung von N_o verzichtet werden, wenn keine Kenntnis über die Migrationsdauer notwendig ist. Ursache ist, dass der Transfer mit den neuesten Werten der Transfermenge beginnt und automatisch endet, wenn der Zustand des Empfängers vollständig gefüllt ist oder das Ende des Fensters des Zustandssenders erreicht ist und somit keine weiteren Werte für den Transfer zur Verfügung stehen.

Für die Bestimmung der tatsächlichen Transferparameter eines einzelnen Fensters ist wie folgt vorzugehen. Entsprechend der ausgewählten Transferstrategie errechnet sich die tatsächliche Anzahl von Zustandstransferwerten N'_o aus Gleichung 5.39 unter Berücksichtigung des vorläufigen Wertes N_o sowie w_o und gegebenenfalls N_c.

$$N'_o = \begin{cases} \min(N_o, w_o - 1), & \text{für OF, LF+, RSD, RSD+, DS, DS+} \\ \min(N_o, w_o - N_c), & \text{für LF} \end{cases} \tag{5.39}$$

Mit N'_o kann anschließend die tatsächliche Anzahl der neuen Werte N'_n berechnet werden, d. h. $N'_n = w_n - N'_o$ (vgl. Gleichung 5.16). Der Wert N'_o ersetzt dann den vorläufigen Transferparameter[11] in der Zustandspaartabelle, wobei sich für Fensterverkleinerungen und gleichgroße Fenster keine Änderungen ergeben.

In Abbildung 5.10 ist ein beispielhafter Verlauf der tatsächlichen Größen jeweils für LF (grau) und OF (schwarz) dargestellt. OF steht dabei auch stellvertretend für die Strategien LF+, RSD, RSD+, DS und DS+. Die Ergebnisse sind für w_n zwischen 0 und 600 angegeben und gelten für eine Konfiguration mit $w_o = 250$, $N_i = 1$ und $N_t = 8$. Die horizontale Linie und die vertikale Linie markieren jeweils w_o. Zudem sind die Vergleichsgrößen $w_{o,lf}$ und $w_{o,of}$ eingezeichnet. Bis zu $w_n = w_o$ unterscheiden sich die Verläufe der Strategien nicht. Für LF ist an dieser Stelle jedoch der maximale Wert für N_o mit 222 Werten erreicht und N_n steigt fortan schneller. Für OF steigt der Wert für N_o weiter und erreicht das Maximum von 249 Werten bei $w_n = 282$. Der Unterschied der Maximalwerte entsteht durch die Zustandswerte, die mit OF übertragen werden, bevor sie verworfen werden. Mit LF ist ihre Übertragung nicht möglich. Aus diesem

[10] In Kapitel 6 wird ein entsprechendes Näherungsverfahren für derartige Anwendungsfälle beschrieben.
[11] Im weiteren Verlauf sind N_o und N_n als tatsächliche Transferparameter zu verstehen.

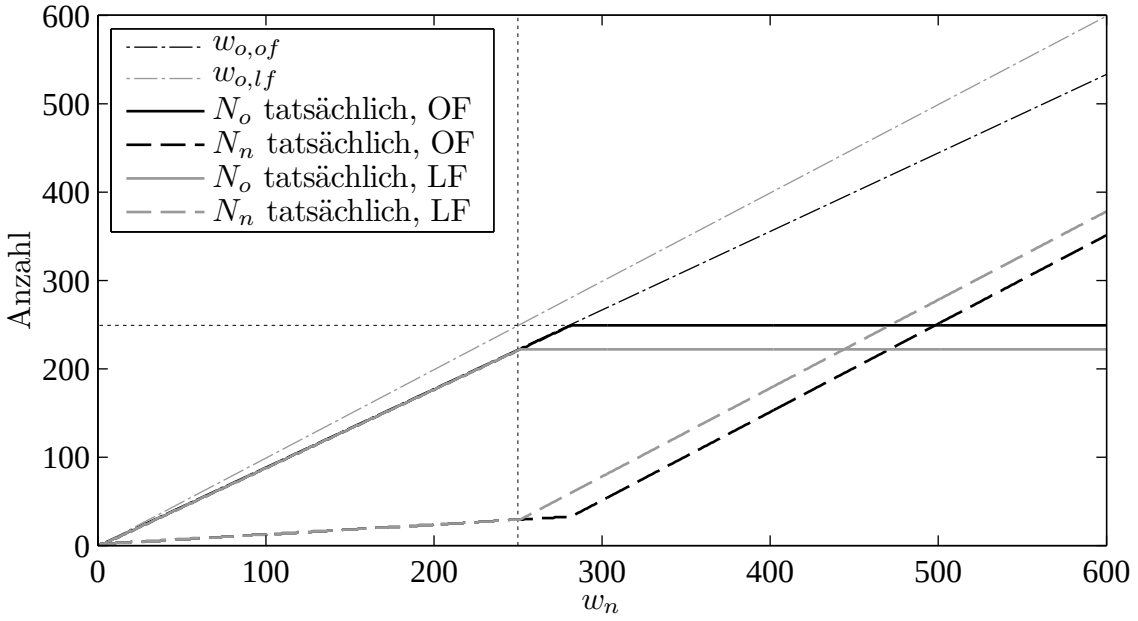


Abbildung 5.10: Parameterabschätzung und tatsächliche Werte von N_o und N_n für die Zustandstransferstrategien OF (LF+, RSD, RSD+, DS, DS+) und LF

Grund werden bei Fenstervergrößerungen mit OF und vergleichbaren Strategien bessere Ergebnisse als mit LF erzielt.

Zusammenfassend bleibt festzuhalten, dass mit OF, LF+, RSD+, DS+ die optimale Migration erreicht werden kann, da mit ihnen Werte übertragen werden können, bevor sie in der Originalanfrage verworfen werden. Bei LF, RS, RSD, DS ist mit Verlusten durch Verwerfen zu rechnen, wodurch sich die Migration verlängert. SemS kann dort, wo es anwendbar ist, die Migrationszeit wesentlich verkürzen. Als Ergebnis der Auswahl der Zustandstransferstrategie wird die Zustandspaartabelle mit dem Strategienamen erweitert und gegebenenfalls wird N_o in der Tabelle an den Wert der tatsächlich übertragbaren Werte angepasst.

5.3.2 Globales Transferverhalten

Das globale Transferverhalten ist für Anfragen mit mehreren Zuständen festzulegen. Beim Transfer von Einzelzuständen sind alle notwendigen Konfigurationsparameter mit der Festlegung der Zustandstransferstrategie vorhanden.

Im Unterschied zum lokalen Transferverhalten, welches die Anzahl und Reihenfolge der zu transferierenden Werte innerhalb jedes Zustands festlegt, wird mit dem globalen Transferverhalten die Verarbeitungsreihenfolge sowie die Anzahl der jeweils zu transferierenden Werte spezifiziert. Die Festlegung dieser Parameter sollte im Zusammenhang mit der Auswahl der Zustandstransferstrategie erfolgen, da dabei ohnehin die

System-, Anfrage- und Zustandseigenschaften analysiert werden. Mögliche Einflussfaktoren, z. B. die Reihenfolge der Eingangsdatenströme oder der Eingangsdatenstrom mit dem die Migration beginnt, wurden bereits in den vorherigen Abschnitten diskutiert.

Zustandssender	Zustandsempfänger	d	N_o	ZTS	Rf.	N_g
$Op_{(AB)}.\mathcal{Z}_A$	$Op_{(AB)}.\mathcal{Z}_A$	2	6	OF	4	3
$Op_{(AB)}.\mathcal{Z}_B$	$Op_{(AB)}.\mathcal{Z}_B$	3	9	OF	1	3
$Op_{((AB)C)}.\mathcal{Z}_C$	$Op_{(CD)}.\mathcal{Z}_C$	0	6	OF	2	3
$Op_{(((AB)C)D)}.\mathcal{Z}_D$	$Op_{(CD)}.\mathcal{Z}_D$	1	6	OF	3	3

Tabelle 5.5: Vervollständigung der Zustandspaartabelle mit den Informationen zur Zustandstransferstrategie (ZTS), Übertragungsreihenfolge (Rf.) und Anzahl pro Transfer (N_g) für ein Szenario mit $w_o \geq 6$, $w_n = 12$, $N_i = 3$, $N_t = 5$ und einem Migrationsbeginn bei Eingangswerten von $\mathcal{S}_C$

Das Resultat ist die Erweiterung der Zustandspaartabelle um die Reihenfolge und die Gruppengröße N_g, d. h. die Anzahl der zu übertragenden Werte pro Verarbeitungsaufruf. Ein Beispiel der endgültigen Zustandspaartabelle repräsentiert Tabelle 5.5. Diese bezieht sich auf das Migrationsbeispiel aus Abbildung 5.7 (S. 92) und vervollständigt Tabelle 5.3 (S. 95). Neben den Konfigurationsparametern für das globale Transferverhalten enthält sie für jeden Zustand die ausgewählte Zustandstransferstrategie (ZTS).

Durch die Verwendung von OF und den Umstand, dass die Migration mit Eingangswerten von $\mathcal{S}_C$ beginnt, kann sogar für $w_o = 6$ die schnellstmögliche Migration realisiert werden. Voraussetzung dafür ist, dass jeweils die drei ältesten Zustandswerte der Zustände $\mathcal{Z}_B$, $\mathcal{Z}_C$ und $\mathcal{Z}_D$ transferiert werden, bevor sie durch Eingangswerte von $\mathcal{S}_A$ verworfen werden. Mit der sich wiederholenden Übertragungsreihenfolge B–C–D–A und einer Anzahl von jeweils drei Datentupeln pro Zustand ist es möglich, diese neun Werte innerhalb von zwei Zyklen zu übertragen. Ebenso hätte die Reihenfolge D–C–B–A verwendet werden können.

Die Abarbeitung der Zustände in Form einer wiederkehrenden Sequenz ist die einfachste Möglichkeit, den Ablauf des Zustandstransfers zu steuern. Ist eine sequenzielle Verarbeitung ungeeignet, kann jede andere Reihenfolge verwendet werden. Für die Suche einer geeigneten Reihenfolge ist stets zu berücksichtigen, wann welche Werte beim Zustandssender verworfen werden.

5.3.3 Migrationsdauer und Zustandstransferdauer

Mit der Festlegung des lokalen und globalen Transferverhaltens kann eine Aussage über die Migrations- und Zustandstransferdauer getroffen werden. Eine weitere relevante Zeitspanne ist die Umschaltdauer. Die drei Zeiten werden nachfolgend im Einzelnen erläutert und sind in Abbildung 5.11 dargestellt. In der Abbildung ist der zeitliche Verlauf

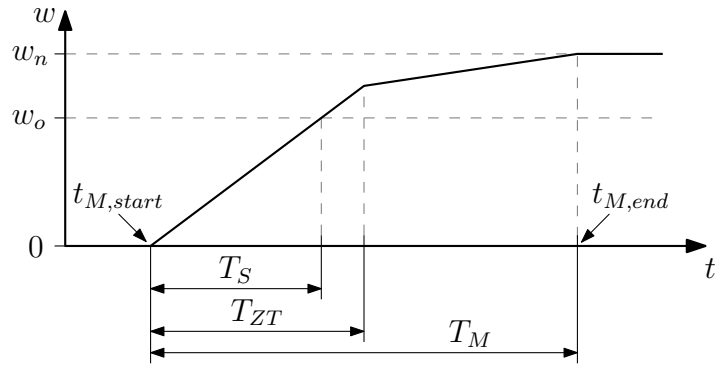


Abbildung 5.11: Qualitative Darstellung des Fensterinhalts des neuen Operators über der Zeit sowie der Größen T_S, T_{ZT} und T_M am Beispiel einer Fenstervergrößerung

der Fenstergröße der Zielanfrage (w) während und nach der Migration am Beispiel einer Fenstervergrößerung qualitativ skizziert. Die Fenstergröße der Originalanfrage (w_o) und die Zielgröße (w_n) sind ebenfalls eingetragen.

Die Migrationsdauer T_M beschreibt die Zeitspanne zwischen Migrationsbeginn $t_{M,start}$ und Migrationsende $t_{M,end}$. Der Migrationsbeginn ist gleichzusetzen mit der Verarbeitung des ersten Wertes des Zustandsempfängers. Dies kann entweder das Entnehmen und Verarbeiten eines Eingangswertes aus der Eingangswarteschlange durch den Zustandsempfänger sein oder, bei Verwendung des sofortigen Zustandstransfers, die Auswahl und der anschließende Transfer eines Wertes aus dem Zustand des Zustandssenders. Das Migrationsende entspricht dem letzten migrationsrelevanten Wert, welcher ebenso ein regulärer Eingangswert oder ein Transferwert sein kann.

Die Zustandstransferdauer T_{ZT} ist die Zeitspanne des Zustandstransfers beginnend mit der Auswahl des ersten Zustandswertes beim Zustandssender. Sie endet, wenn der letzte Transferwert beim Zustandsempfänger verarbeitet und in den Zustand eingefügt wurde.

Die Umschaltdauer T_S ist die Zeit, bis zur neuen Anfrage umgeschaltet wird[12]. Diese Zeitspanne beginnt in jedem Fall mit $t_{M,start}$. Ihre Länge ist von der Umschaltbedingung abhängig, z. B. ob zur neuen Anfrage gewechselt wird, wenn das neue Fenster vollständig gefüllt ist ($w = w_n$) oder bereits wenn das neue Fenster mehr Werte enthält als das alte ($w > w_o$). Im ersten Fall ist $T_S = T_M$. Der zweite Fall, dargestellt in Abbildung 5.11, kann nur bei Fenstervergrößerungen ($w_n > w_o$) auftreten. Die Umschaltdauer T_S gibt demnach an, wie viel Zeit ab Migrationsbeginn vergeht, bis die Ergebnisse von der neuen Anfrage ausgegeben werden und nicht mehr von der alten.

Für das Beispiel der Fenstervergrößerung in Abbildung 5.11 ist $T_S < T_{ZT} < T_M$. Zur neuen Anfrage wird gewechselt, wenn w die Originalfenstergröße w_o übersteigt.

[12] nicht nur der Umschaltvorgang

Die Migration und der Zustandstransfer werden fortgeführt. Nach dem Abschluss des Zustandstransfers wächst die Fenstergröße langsamer, da nur noch Eingangswerte verarbeitet werden. Mit Erreichen der neuen Fenstergröße w_n ist die Migration abgeschlossen.

Für alle Zustandstransferstrategien gilt immer $T_{ZT} \leq T_M$ und $T_S \leq T_M$. Da Zustandstransfer und Umschaltzeit unabhängig voneinander sind, kann im Unterschied zum dargestellten Beispiel auch $T_{ZT} \leq T_S$ gelten. Insbesondere bei SemS besteht die Möglichkeit, sehr zeitig zur neuen Anfrage umzuschalten, d. h. $T_S \ll T_{ZT} \leq T_M$. Allerdings kann für SemS keine allgemeine Berechnungsvorschrift für die drei Größen angegeben werden, da diese vom Einzelfall abhängen. Aus diesem Grund wird SemS von der Betrachtung ausgeschlossen. Für alle anderen Zustandstransferstrategien sind T_{ZT} und T_M anhand der Informationen aus der Zustandspaartabelle berechenbar. Für T_S ist zudem die Umschaltbedingung zu berücksichtigen. Die Betrachtung erfolgt zunächst für Anfragen mit einem Zustand und anschließend für Anfragen mit mehreren Zuständen.

Anfragen mit einem Zustand Für die Migration von Anfragen mit einem Zustand wurden in Abschnitt 5.2.1 Formeln für die Berechnung von zwei beispielhaften Fällen (a) und (b) vorgestellt. Dabei wurde die Berechnung für beide Fälle so ausgelegt, dass die Migration mit einem Wert des Eingangsdatenstroms abgeschlossen wird. Der Beginn der Migration ist entweder ein Eingangswert oder ein Zustandswert falls sofort mit dem Zustandstransfer begonnen wird. Für den Zeitraum des anfänglichen Zustandstransfers wurde in Abschnitt 5.2.1 die Laufzeitgröße T_a definiert. Die verbleibende Zeit der Migration erstreckt sich zwischen dem ersten und dem letzten Eingangswert der Migration. Somit ist die Migrationsdauer T_M allein von N_n und T_a abhängig. Alle anderen Größen sind durch das Anfrage- und Systemverhalten vorgegeben. Mittels N_n^*, der Anzahl von Eingangswerten im letzten Zyklus (bei $N_i > 1$), kann T_M berechnet werden. Wird der sofortige Zustandstransfer nicht verwendet, ist $T_a = 0$ und der letzte Summand der Gleichung (5.41) entfällt.

$$N_n^* = ((N_n - 1) \mod N_i) + 1 \tag{5.40}$$

$$T_M = \left\lfloor \frac{N_n - 1}{N_i} \right\rfloor * T + N_n^* * T_p + T_a \tag{5.41}$$

Die Zustandstransferdauer T_{ZT} ist von der Anzahl der transferierten Zustandswerte N_o abhängig. Beim Einsatz des sofortigen Zustandstransfers ist zudem zu beachten, wie viele Zustandswerte während der ersten Transferzeit, noch vor Eintreffen des ersten Eingangswertes, übertragen werden. Diese Anzahl ist als N_a unter den Konfigurationsparametern abgelegt. Folglich lässt sich die Anzahl der Transferwerte während der letzten Transferzeit (N_o^*) entsprechend der bereits definierten Gleichung (5.19) errechnen. Diese sei aus Gründen der Anschaulichkeit hier wiederholt.

$$N_o^* = ((N_o - N_a - 1) \mod N_t) + 1$$

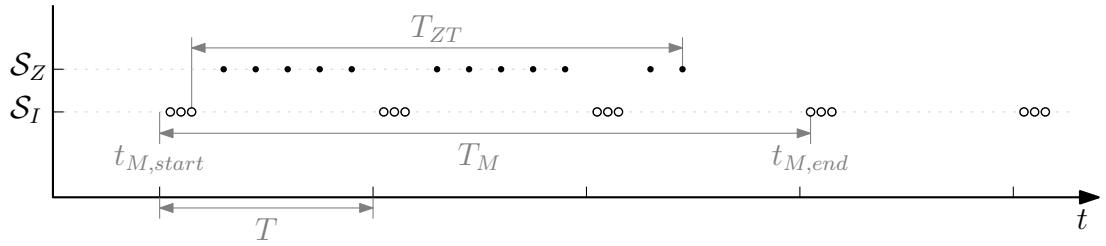


Abbildung 5.12: Migrations- und Zustandstransferdauer für ein einzelnes Fenster mit $N_i = 3$, $N_t = 5$ und $w_n = 22$ (entspricht Fall (a) in Abbildung 5.3)

Falls kein sofortiger Zustandstransfer angewandt wird, ist N_o^* $(N_a = 0)$ zu benutzen. Die Zustandstransferdauer hängt dann ausschließlich von N_o ab. Wird der sofortige Zustandstransfer eingesetzt, kommen zwei Zeiten hinzu. Zum einen die Zeit für den anfänglichen Zustandstransfer T_a und außerdem die Zeit für die Verarbeitung der ersten Eingangsdaten $T_p^* = N_i * T_p$, welche dann in den Zeitraum des Zustandstransfers fällt. Die Gleichung zur Berechnung der Zustandstransferdauer ohne und mit sofortigem Zustandstransfer (SZT) lautet:

$$T_{ZT} = \left\lfloor \frac{N_o - N_a - 1}{N_t} \right\rfloor * T + N_o^* * (T_t + T_p) + \begin{cases} 0\,, & \text{ohne SZT } (N_a = 0) \\ T_a + T_p^*\,, & \text{mit SZT} \end{cases} \tag{5.42}$$

In Abbildung 5.12 sind die Migrations- und Zustandstransferdauer für ein einzelnes Fenster mit $N_i = 3$, $N_t = 5$ und $w_n = 22$ dargestellt. Das Beispiel entspricht Fall (a) in Abbildung 5.3 (S. 85). Ein sofortiger Zustandstransfer wird nicht durchgeführt, d. h. $T_a = 0$ und $N_a = 0$. Die Migration beginnt und endet jeweils mit einem Wert des Eingangsdatenstroms $\mathcal{S}_I$. Die Migrationsdauer beträgt $T_M = 3 * T + T_p$. Der Zustandstransfer beginnt nach der Bearbeitung der ersten Eingangswerte. Die Zustandstransferdauer umfasst die Verarbeitung aller Werte des Zustandstransferdatenstroms $\mathcal{S}_Z$ und beträgt $T_{ZT} = 2 * T + 2 * (T_t + T_p)$.

Anfragen mit mehreren Zuständen Für die Migration von Anfragen mit mehreren Zuständen beschränkt sich die Betrachtung nicht nur auf die im Abschnitt 5.2.2 vorgestellten Migrationsfälle (a) und (b), sondern kann vielmehr auf jede Migration mit homogenen Zuständen angewendet werden, welche dem beschriebenen Prinzip entspricht, d. h. ein (schwankungsbeschränktes) periodisches Systemverhalten und Nutzung der Pausendauer zur Zustandsübertragung. Die Berechnung der Zustandstransfer- und Migrationsdauer bezieht sich auf die gesamte Anfrage und nicht auf einzelne Zustände, da die Migrationsparameter zwar für die Zustände bestimmt wurden, die Migration jedoch für die gesamte Anfrage optimiert ist.

Aus der Zustandspaartabelle ist für jeden Zustand jeweils die Anzahl der neuen Werte (N_n) und der Zustandstransferwerte (N_o) bekannt. Zudem existiert gegebenenfalls eine

Angabe, wie viele Werte jedes Zustands durch sofortigen Zustandstransfer übertragen werden (N_a). Für die Migration der gesamten Anfrage kann daraus jeweils die Summe von neuen Werten ($\mathcal{N}_n$), Zustandstransferwerten ($\mathcal{N}_o$) und anfänglichen Transferwerten ($\mathcal{N}_a$) über alle N_z Zustände bestimmt werden. Die Berechnung der Summen ist in den nachfolgenden Gleichungen (5.43) aufgeführt.

$$\text{(a)} \quad \mathcal{N}_n = \sum_{i=1}^{N_z} N_{n,i} \qquad \text{(b)} \quad \mathcal{N}_o = \sum_{i=1}^{N_z} N_{o,i} \qquad \text{(c)} \quad \mathcal{N}_a = \sum_{i=1}^{N_z} N_{a,i} \tag{5.43}$$

Die Anzahl der letzten Eingangswerte kann analog Gleichung (5.40) bestimmt werden mit dem Unterschied, dass $\mathcal{N}_n$ entsprechend Gleichung (5.43a) als Ausgangsgröße genutzt wird. Die Formel lautet somit

$$N_n^* = ((\mathcal{N}_n - 1) \mod N_i) + 1. \tag{5.44}$$

Daraus lässt sich die Migrationsdauer bestimmen. Die Berücksichtigung des Systemverhaltens (Variante (a) oder (b)) erfolgt durch T_{syst}.

$$T_M = \left\lfloor \frac{\mathcal{N}_n - 1}{N_i} \right\rfloor * T_{syst} + N_n^* * T_p + T_a \tag{5.45}$$

Für die Berechnung der Zustandstransferdauer ist zunächst N_o^* zu ermitteln, hier unter Verwendung der Summe $\mathcal{N}_o$ nach Gleichung (5.43b). Anschließend kann die Zustandstransferdauer mit Berücksichtigung der Systemvariante ermittelt werden. Falls ein sofortiger Zustandstransfer eingesetzt wird, ist $\mathcal{N}_a$ gemäß (5.43c) zu benutzen.

$$N_o^* = ((\mathcal{N}_o - \mathcal{N}_a - 1) \mod N_t) + 1 \tag{5.46}$$

$$T_{ZT} = \left\lfloor \frac{\mathcal{N}_o - \mathcal{N}_a - 1}{N_t} \right\rfloor * T_{syst} + N_o^* * (T_t + T_p) + \begin{cases} 0\,, & \text{ohne SZT } (\mathcal{N}_a = 0) \\ T_a + T_p^*\,, & \text{mit SZT} \end{cases} \tag{5.47}$$

Zusammenfassung Migrations-, Zustandstransfer- und Umschaltdauer können bereits während der Planung berechnet werden, um Alternativen zu vergleichen und das optimale Migrationsverhalten festzulegen. Aus den Migrationsparametern (N_n, N_o, N_a, N_t, N_z), den Systemeigenschaften (T_{syst}, T_p, T_t, N_i) und Laufzeitgrößen (T_a) lassen sich Migrationsdauer und Zustandstransferdauer für Anfragen mit homogenen Zuständen in periodischen Systemen vorhersagen. Die Umschaltdauer ist ebenso berechenbar, hängt aber zusätzlich von der festgelegten Umschaltbedingung ab. Es wird deutlich, dass die Bestimmung der Migrations- und Zustandstransferdauer für Anfragen mit einzelnen Zuständen einen Sonderfall der Gleichungen (5.43) bis (5.47) darstellt und mit $N_z = 1$ und $T_{syst} = T$ ebenso durch diese Gleichungen berechnet werden kann.

Die vorgestellten Gleichungen sind für alle Zustandstransferstrategien außer SemS einsetzbar. Da für SemS die Größen N_n und N_o unter Umständen nicht festgelegt werden, sondern sich zur Laufzeit durch die Auswahl der relevanten Werte ergeben, kann

eine Vorhersage von T_M, T_{ZT} und T_S nicht auf Basis einer allgemeinen Berechnungsvorschrift erfolgen. Stattdessen ist der Einzelfall gemäß seiner Migrationseigenschaften zu bewerten.

5.4 Durchführung

Die Abschnitte der Durchführung – funktionale Erweiterungen, Zustandstransfer und Abschluss (gemäß Abbildung 5.2) – werden zusammenhängend betrachtet und in erster Linie für die Migration von Anfragen mit mehreren Zuständen in verteilten, periodischen Systemen. Falls Vereinfachungen für die Migration von Einzelzuständen möglich sind, werden diese angegeben. Funktionale Erweiterungen werden für Operatoren und für das gesamte System beschrieben.

Eine der Hauptanforderungen war, dass der Zustandstransfer den operativen Betrieb nicht beeinflussen darf. Deshalb ist der Zustandstransfer ausschließlich während der Pausen durchzuführen. Die funktionalen Erweiterungen dienen dazu, diese Anforderung zu realisieren. Mit Hilfe der vorangegangenen Schritte wurde eine geeignete Konfiguration für die Migration erstellt und in einer Zustandspaartabelle gespeichert. Diese Tabelle sowie der Parameter N_t sind die Basis aller durchzuführenden Erweiterungen. Die Steuerung des Zustandstransfers wird von einem Update-Manager übernommen, d. h. ein zusätzliches Systemelement, welches den Transfer koordiniert.

5.4.1 Funktionale Erweiterungen von Operatoren

Aus der Zustandspaartabelle gehen Zustandssender und Zustandsempfänger hervor. Somit steht fest, welche Operatoren mit zusätzlicher Funktionalität erweitert und wie die Operatoren verbunden werden müssen. Entsprechend ihrer Rolle sind die Operatoren mit unterschiedlichen Funktionen zu versehen.

Zustandssender werden für jeden zu übertragenden Zustand mit einer Transferfunktion erweitert. Diese Funktion realisiert die spezifizierte Zustandstransferstrategie, welche auf der Transfermenge, der ausgewählten Untermenge des Gesamtzustandes, angewendet wird. In Abbildung 5.13 ist ein Beispiel für einen entsprechenden Operator in der Rolle eines Zustandssenders dargestellt. Jeder Zustand besitzt eine eigene Transferfunktion mit der Zustandstransferstrategie (ZTS_A bzw. ZTS_B).

Die Ausführung der Zustandstransferstrategien erfolgt im Wechsel mit der Verarbeitung der Eingangswerte. Da der Zustandstransfer eine geringere Priorität besitzt, werden zunächst alle Datentupel der Eingangswarteschlangen verarbeitet. Danach signalisiert der Update-Manager der jeweiligen Zustandstransferstrategie, wie viele Werte aus der Transfermenge zu übertragen sind. Die ausgewählten Werte werden über zeitweilig verfügbare Ausgangsdatenströme, im Beispiel die Ausgangsdatenströme $\mathcal{S}_{Z_A}$ bzw. $\mathcal{S}_{Z_B}$, an die entsprechenden Zustandsempfänger gesendet. Nach Abschluss des Transfers pausiert

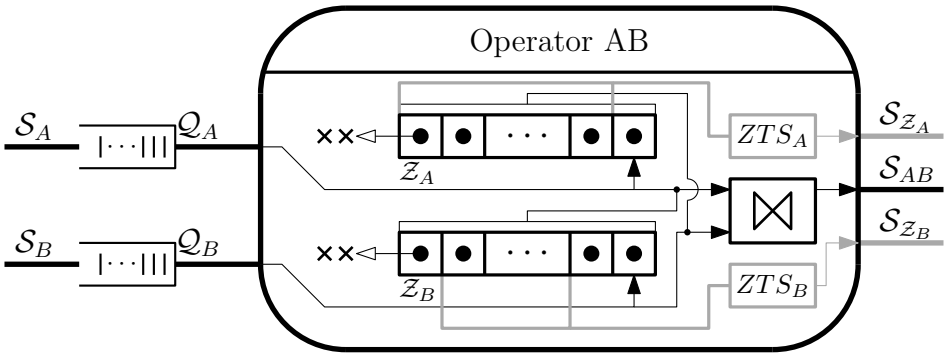


Abbildung 5.13: Zustandssender mit zusätzlichen, temporären Ausgangsdatenströmen und Funktionen zur Umsetzung der Zustandstransferstrategien (grau)

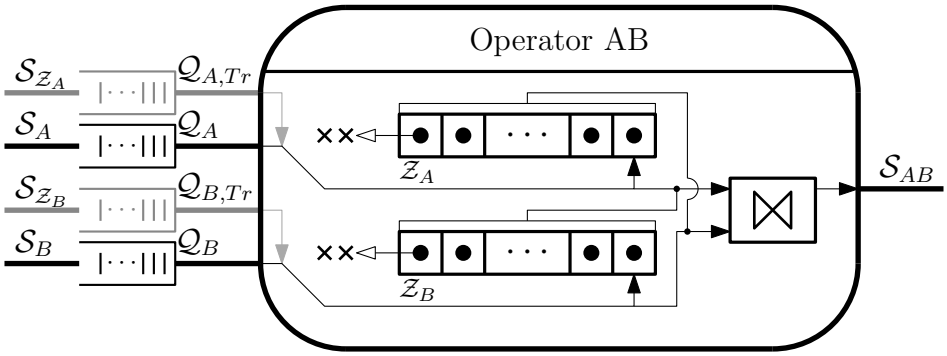


Abbildung 5.14: Zustandsempfänger mit zusätzlichen, temporären Eingangswarteschlangen (grau)

der Operator, bis neue Eingangswerte zu verarbeiten sind oder der Update-Manager den nächsten Transfer beauftragt.

Für das Aufnehmen der transferierten Zustandswerte bieten Zustandsempfänger temporär zusätzliche Warteschlangen an, sogenannte Transferwarteschlangen. Dies dient hauptsächlich der Unterscheidung von Transfertupeln und regulären Datentupeln, wodurch reguläre Tupel priorisiert und Transfertupel gegebenenfalls speziell behandelt werden können. Ein Beispiel für einen Zustandsempfänger ist in Abbildung 5.14 dargestellt. Der Verbundoperator besitzt für den Zeitraum des Transfers die Transferwarteschlangen $\mathcal{Q}_{A,Tr}$ und $\mathcal{Q}_{B,Tr}$. Diese erfassen die Werte der Eingangsdatenströme $\mathcal{S}_{\mathcal{Z}_A}$ bzw. $\mathcal{S}_{\mathcal{Z}_B}$, welche von den Zustandssendern generiert werden. Funktionell unterscheiden sich Transferwarteschlangen von den normalen Warteschlangen nicht. Auch die Verbindung zwischen Zustandsempfänger und Zustandssender entspricht dem Publish-Subscribe-Mechanismus, der genutzt wird, um Systemelemente miteinander zu verbinden.

Datentupel aus einer Transferwarteschlange werden mit einer geringeren Priorität als reguläre Werte verarbeitet, d. h. wann immer Daten in den normalen Warteschlangen

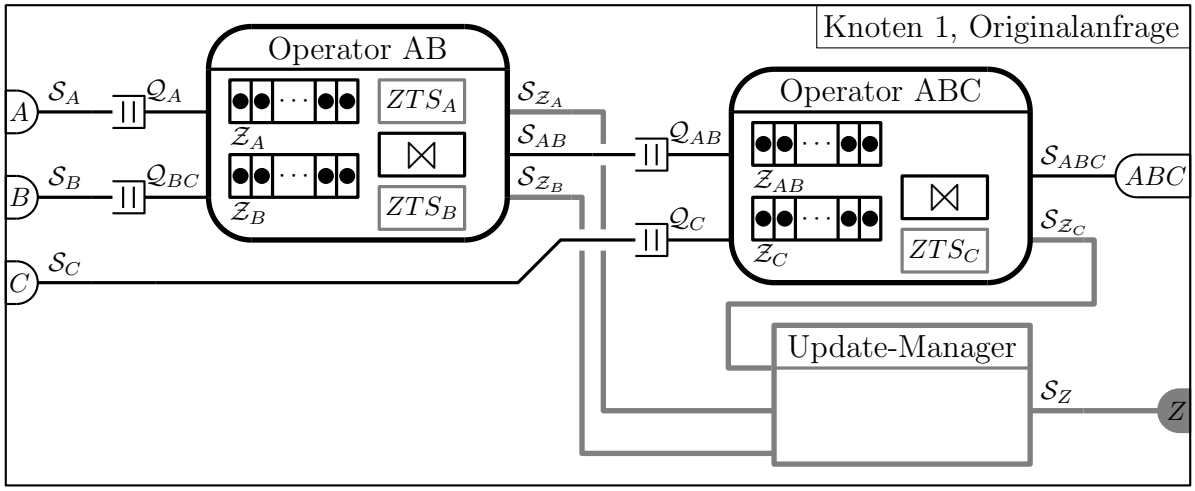


Abbildung 5.15: Datenfluss eines Verarbeitungsknotens mit Zustandssendern und Update-Manager

vorliegen, werden diese zuerst bearbeitet, egal wie viele Werte in den Transferwarteschlange vorhanden sind. Die innere Verarbeitung der Transferwerte nutzt im Wesentlichen die gleichen Funktionen wie die normale Verarbeitung, Zustände werden bereinigt oder mit den neuen Werten ergänzt und Ergebnisse werden auf Basis des eingestellten Algorithmus erzeugt. Zusätzlich kann eine Sortierfunktion bei der Aktualisierung des Zustandes eingesetzt werden. Außerdem besteht die Möglichkeit, für Transfertupel die Ergebnisausgabe zu unterdrücken. Ist es nötig, für Transferwerte einen modifizierten Algorithmus zu verwenden, kann dieser für die Dauer des Zustandstransfers vom Operator genutzt und abschließend durch den neuen Algorithmus ersetzt werden.

Nach Abschluss des Zustandstransfers werden alle Zusatzfunktionen, d. h. Zustandstransferstrategien, zusätzliche Ein- oder Ausgangsdatenströme, Transferwarteschlangen, Sortieralgorithmen usw., aus den Operatoren entfernt. Dadurch werden Mehrkosten bezüglich CPU-Zeit und Speicher vermieden, und eine normale Verarbeitung findet statt. Bei einem notwendigen Wechsel des Verarbeitungsalgorithmus wird die Algorithmusaustauschfunktionalität (Abschnitt 4.2.2) genutzt.

5.4.2 Funktionale Erweiterungen des Datenstromsystems

Auf jedem Verarbeitungsknoten, der mindestens einen Zustandssender oder -empfänger hat, wird ein Update-Manager angelegt, der u. a. die Fähigkeiten besitzt, wie ein zusätzliches Systemelement zu agieren. Ein Update-Manager realisiert das globale Transferverhalten und übernimmt im Zusammenwirken mit Zustandssendern oder Zustandsempfängern unterschiedliche Aufgaben.

Abbildung 5.15 zeigt den Datenfluss auf einem Verarbeitungsknoten mit Zustandssendern und einem Update-Manager. Der Kontrollfluss ist nicht dargestellt. Die Aufgabe

des Update-Managers ist, die Transferfunktionen der Operatoren in der konfigurierten Reihenfolge aufzurufen und deren Ergebnisse an den Update-Manager des Zustandsempfängers zu senden. Beim Aufruf wird einer Transferfunktion die Anzahl von Werten genannt, die übertragen werden soll. Die Informationen bezieht der Update-Manager aus der Zustandspaartabelle und aus der Beobachtung des Systems. Der Update-Manager hat zu entscheiden, wann der Zustandstransfer beginnen kann, welcher Zustand als nächstes übertragen wird und wie viele Werte dieses Zustands übertragen werden.

Um den korrekten Zeitpunkt zu kennen, wann der Zustandstransfer begonnen oder fortgesetzt werden kann, benötigt der Update-Manager die Kenntnis, dass alle Eingangswerte verarbeitet wurden. Für einen einfachen Entscheidungsmechanismus kann die Datensenke mit einer Schnittstelle versehen werden, die deren Betriebszustand angibt. Solange sich Werte in der Eingangswarteschlange der Datensenke befinden und diese verarbeitet werden, ist der Betriebszustand „aktiv“, ansonsten „ruhend“. Signalisiert die Datensenke, dass sie ihre Verarbeitung abgeschlossen hat, indem sie ihren Betriebszustand auf „ruhend“ ändert, wird der Update-Manager benachrichtigt und fährt mit dem Verlauf des Zustandstransfers fort. Um die Zuverlässigkeit zu erhöhen, sollten alle Systemelemente eine solche Schnittstelle implementieren. In diesem Fall löst der Update-Manager den Zustandstransfer erst dann aus, wenn alle Systemelemente eines Knotens den Betriebszustand „ruhend“ eingenommen haben. Soll das System auch die Migration von Teilanfragen unterstützen ist eine solche Funktion ohnehin unentbehrlich.

Die Reihenfolge, mit der die einzelnen Zustandstransferstrategien aufgerufen werden sollen, und die Anzahl, die pro Aufruf übertragen werden soll (N_g), kennt der Update-Manager aus der Zustandspaartabelle. Mit N_t ist dem Update-Manager bekannt, wie viele Werte während einer Pause übertragen werden können. Entsprechend der Reihenfolge ruft der Update-Manager die Transferfunktion eines Operators auf und übergibt die Anzahl der zu übertragenden Werte. Ist der Transfer eines Zustandes beendet, wird die nächste Transferfunktion durch den Update-Manager aufgerufen und die entsprechende Anzahl übergeben. Für die Bestimmung der Anzahl sind zwei Sachverhalte zu berücksichtigen. Wie viele Werte müssen für diesen Zustand noch übertragen werden? Wie viele Werte können in dieser Pause noch übertragen werden?

In der Zustandspaartabelle ist die Zahl der insgesamt zu übertragenden Werte als N_o hinterlegt. Sie wird zu Beginn der Migration für jeden Zustand in $N_{o,r}$ gespeichert. Für jeden übertragenen Wert wird $N_{o,r}$ um eins reduziert, bis alle Werte übertragen wurden. Das jeweilige Zustandspaar wird dann aus der Zustandspaartabelle entfernt. Die Anzahl, die für einen Zustand pro Aufruf übertragen werden soll ist $N = \min(N_{o,r}, N_g)$. Ob diese Anzahl übertragen werden kann, hängt ab von N_p, den verbleibenden Werten während einer Pause. Ausgangsgröße für die Bestimmung ist N_t. Zu Beginn einer Pause wird $N_p = N_t$ gesetzt. Jeder übertragene Wert, unabhängig von welchem Zustand, wird von N_p abgezogen, bis $N_p = 0$ ist. Für den nächsten aufzurufenden Zustand ist die Anzahl der tatsächlich zu übertragenden Datentupel $\min(N, N_p)$. Nur beim letzten Zustand einer Pause kann es vorkommen, dass $N_p < N$. Gemäß der Bedingung werden dann N_p Tupel übertragen und danach ist der Zustandstransfer für diese Pause abgeschlossen. Der

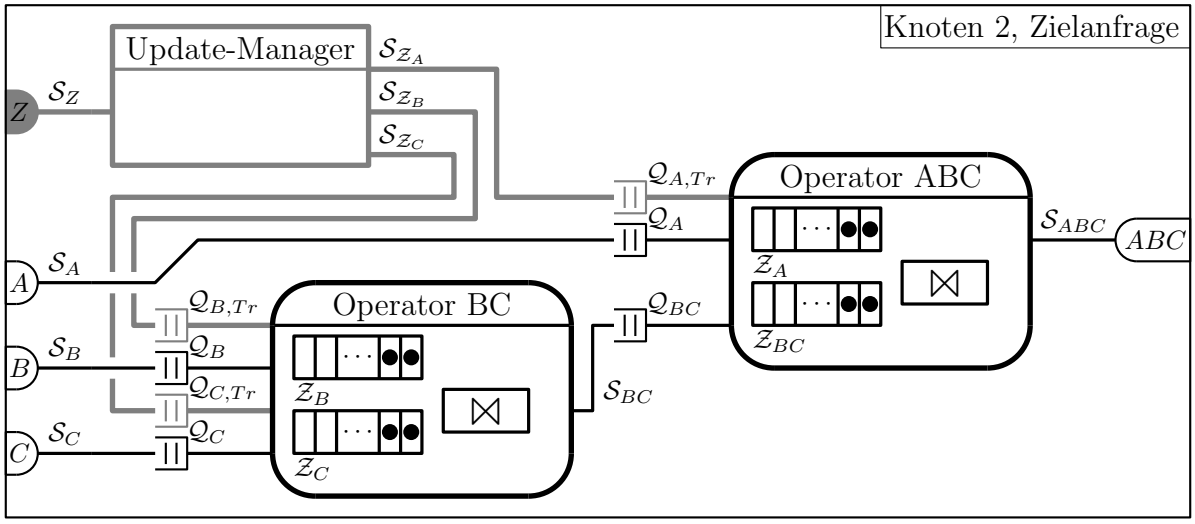


Abbildung 5.16: Datenfluss eines Verarbeitungsknotens mit Zustandsempfängern und Update-Manager

Verarbeitungsknoten wartet dann auf die nächsten Eingangsdaten und verarbeitet diese. Der Zustandstransfer wird in der nächsten Pause mit der Übertragung der verbleibenden $N_p - N$ Werte des letzten Zustands fortgesetzt. Danach wird der nächste Zustand in der Reihenfolge ausgewählt usw.

Alle Werte, die von einer Zustandstransferfunktion ausgewählt wurden, werden dem Update-Manager übergeben. Er schickt diese als vereinten Datenstrom an den Update-Manager des Zustandsempfängers, im Beispiel $\mathcal{S}_Z$. Dies ermöglicht dem Update-Manager außerdem, den Fortschritt des Zustandstransfers zu verfolgen sowie $N_{o,r}$ und N_p fortlaufend zu aktualisieren.

Auf dem Knoten der Zustandsempfänger werden die empfangenen Daten vom Update-Manager zerlegt und in die entsprechenden Transferwarteschlangen eingefügt. Ein Beispiel ist in Abbildung 5.16 dargestellt. Damit der Update-Manager die Werte den Empfängern zuordnen kann, müssen die einzelnen Werte durch den Update-Manager der Zustandssender mit einer geeigneten Information versehen worden sein.

Sofern sich mehrere Operatoren auf dem Knoten befinden, ist eine Koordination der Empfänger notwendig. Diese Aufgabe übernimmt der Update-Manager. Beispielsweise kann das Einfügen in die Transferwarteschlangen so lange verzögert werden, bis alle Systemelemente des Knotens die reguläre Verarbeitung abgeschlossen haben, sich also im Betriebszustand „ruhend“ befinden. Die Verarbeitung im Operator selbst muss nicht explizit koordiniert werden, da die Transferwarteschlangen mit einer geringeren Priorität im Vergleich zu den regulären Warteschlangen behandelt werden.

Nach Abschluss des Zustandstransfers werden die Update-Manager von den Bearbeitungsknoten entfernt. Dadurch und durch den Rückbau der Operatoren werden alle zusätzlichen Funktionen und Verbindungen, die für den Zustandstransfer erstellt wurden,

beseitigt. Die modifizierte Anfrage entspricht dann vollständig der Zielanfrage.

5.4.3 Zusammenfassung Durchführung

Für die Durchführung des Zustandstransfers sind die beteiligten Operatoren temporär mit zusätzlichen Funktionen zu erweitern. Auf jedem beteiligten Verarbeitungsknoten wird für den Zeitraum des Zustandstransfers ein Update-Manager erzeugt, der die Durchführung des Zustandstransfers anhand der Transferkonfiguration (u. a. Zustandspaartabelle) steuert. Abschließend werden einige Vereinfachungen und mögliche Erweiterungen diskutiert.

In existierenden DSMS wird die Verarbeitung von Anfragen meistens durch einen Scheduler gesteuert. Ein Update-Manager hat im Prinzip die gleiche Aufgabe, nur dass die von ihm ausgelösten Aktivitäten mit einer untergeordneten Priorität ausgeführt werden. Daher ist es in Datenstromsystemen mit Schedulern sinnvoll, die Funktionalität des Update-Managers im Scheduler zu implementieren.

In einem verteilten, periodischen System mit mehreren Transferzuständen wird die Steuerung vollständig von den Update-Managern übernommen. Beim Transfer eines einzelnen Zustandes könnte der Zustandstransfer auch innerhalb der Transferstrategie gesteuert werden. Die Kenntnis von N_t reicht dafür aus. Für die Bestimmung des jeweiligen Startzeitpunktes und das anschließende Auslösen der Zustandstransfers kann weiterhin ein Update-Manager verwendet werden oder die Transferfunktion des Operators besitzt eine Möglichkeit, das System zu beobachten. Aus Gründen der Einfachheit ist es jedoch günstiger, den allgemeinen Ansatz zu verfolgen, da ansonsten Erweiterungen der Transferfunktion erforderlich sind, die nur für diesen speziellen Fall angewendet werden können. Zudem ist das Einsparpotential bezüglich der Systemressourcen gering, da maximal die Kommunikation zwischen Update-Manager und Transferfunktion entfällt.

Wird die Migration auf einem Verarbeitungsknoten[13] durchgeführt, lassen sich die Ausgänge der Zustandssender direkt mit den Transferwarteschlangen der Zustandsempfänger verbinden. Abbildung 5.17 zeigt eine solche Konfiguration für das zuvor verwendete Beispiel. Der Update-Manager übernimmt die Koordination des globalen Transferverhaltens. Da die Datentupel nicht über ihn übertragen werden, benötigt er eine Rückmeldung von den Operatoren, wann sie ihre Zustandstransferaktivitäten abgeschlossen haben. Im einfachsten Fall reicht die Beobachtung der Betriebszustände der Operatoren aus. Die Realisierung des Zustandstransfers für solch ein zentralisiertes System ist zwar deutlich schneller als der Transfer über ein Netzwerk, da der Transfer im Arbeitsspeicher abläuft. Trotzdem sollte bei größeren Transfermengen auf eine Synchronisation nicht verzichtet werden. Eine Synchronisierung ist notwendig, wenn die gesamte Transferzeit die verfügbare Pausenzeit übersteigt.

[13] Es besteht auch die Möglichkeit, den Zustand direkt zwischen die Operatoren zu kopieren oder durch „verteilte Zustände“ für die neuen Operatoren zugänglich zu machen, sofern die Operatoren geeignete Schnittstellen dafür anbieten. Diese Vorgehensweise ist bereits von existierenden Systemen bekannt und wird deshalb hier nicht weiter diskutiert.

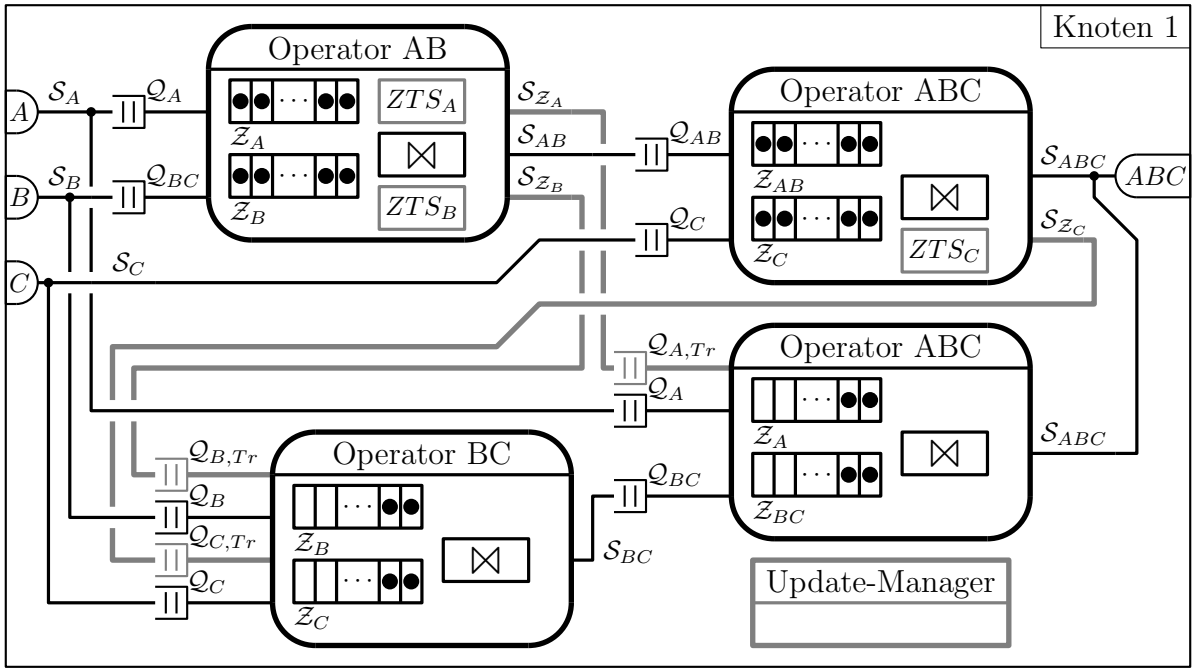


Abbildung 5.17: Zustandstransfer innerhalb eines Verarbeitungsknotens

Bei der Anwendung der vorgestellten Ansätze zur Durchführung des Zustandstransfers in periodischen Systemen wird durch die Konfiguration dafür gesorgt, dass weniger Zeit für die Übertragung genutzt wird als in einer Pause zur Verfügung steht. In aperiodischen Systemen gibt es diese Bedingung nicht, da die Länge der Pausen nicht vorhersagbar ist. Die Grundmechanismen, z. B. Transferwarteschlange, Update-Manager oder Zustandspaartabellen, können trotzdem in aperiodischen Systemen angewendet werden. Funktionen, wie die temporären Transferfunktionen der Zustandssender, müssen aber mit geeigneten Konzepten erweitert werden, um die Anforderungen zu erfüllen. Die Erläuterung solcher Konzepte erfolgt im nächsten Kapitel.

5.5 Zusammenfassung

Der Zustandstransfer wird hauptsächlich während der Reintegration angewendet. Dieses Kapitel beschreibt die Konfiguration und Durchführung des Zustandstransfers für periodische und schwankungsbeschränkte periodische Datenströme. Für ein periodisches System lässt sich das Verhalten genau vorhersagen und deshalb der Zustandstransfer geeignet konfigurieren. Um die Anforderung zu erfüllen, den operativen Betrieb nicht zu beeinflussen, sieht der allgemeine Ansatz vor, den Zustandstransfer während der ansonsten nicht genutzten Pausenzeit durchzuführen. Für das Einhalten der Anforderung ist damit zu gewährleisten, dass die Transferzeit die Pausendauer nicht übersteigt.

Das Ziel der Konfigurationsschritte ist die Bestimmung von geeigneten Konfigurationsparametern zum Erreichen einer optimalen Migration. Die optimale Migration erlaubt die schnellstmögliche Ausgabe von korrekten Ergebnissen durch die Zielanfrage bei den gegebenen Randbedingungen. Die Konfigurationsparameter beschreiben das lokale und das globale Transferverhalten, d. h. das Verhalten innerhalb der beteiligten Operatoren bzw. das Verhalten des gesamten Systems während des Zustandstransfers und der Migration.

Dafür werden zunächst relevante Zustandspaare identifiziert. Sofern Schema und Semantik der Zustände in Original- und Zielanfrage übereinstimmen, ist diese Aufgabe einfach. Besitzen die potentiellen Zustandssender und Zustandsempfänger und deren Anfragen jedoch unterschiedliche Eigenschaften, ist das Finden von geeigneten Paaren deutlich schwieriger, insbesondere, wenn diese Aufgabe automatisiert durchgeführt werden soll. Das Resultat der Zustandspaarsuche ist die Zustandspaartabelle, eine Tabelle aller Zustandssender-Zustandsempfänger-Paare, die für den Zustandstransfer in Frage kommen. Die Zustandspaartabelle wird während der folgenden Schritte fortlaufend aktualisiert und schließlich für die Konfiguration der Migration verwendet.

Stehen die Zustandspaare fest, wird für jeden Zustand die ideale Transfermenge ermittelt. Die Anzahl der tatsächlich zur Verfügung stehend Datentupel wird vorerst nicht berücksichtigt. Die Berechnung der Transfermenge erfordert die Berücksichtigung der Eigenschaften von Zuständen, Operatoren, Anfragen und des Systems. Für den Zustandstransfer in Anfragen mit nur einem Zustand wurde ein Berechnungsmodell für zwei unterschiedliche Systemvarianten vorgestellt, mit dessen Hilfe sich die ideale Transfermenge bestimmen lässt.

Befinden sich mehrere Zustände in der zu migrierenden Anfrage, gestaltet sich die Berechnung deutlich schwieriger, da der Umfang der Eigenschaften und mögliche Abhängigkeiten fast immer eine Betrachtung des Einzelfalls erfordern. Das Berechnungsmodell wurde für die Anwendung auf Anfragen mit mehreren homogenen, unabhängigen Zuständen erweitert. Für Anfragen mit mehreren heterogenen oder abhängigen Zuständen, sei auf den im nächsten Kapitel vorgestellten Ansatz verwiesen. Allerdings wird bereits hier deutlich, dass es für heterogene Zustände äußerst schwierig ist, eine allgemeine Berechnungsvorschrift für die Konfiguration der optimalen Migration zu entwickeln, da Abhängigkeiten und zahlreiche Einflussgrößen sich darauf auswirken. Das Ergebnis der Zustandsauswahl ist die Erweiterung der Zustandspaartabelle um potentielle Transferparameter, u. a. die Anzahl der zu übertragenden Werte. Diese Anzahl entspricht der Größe der potentiellen Transfermenge und stellt einen Idealwert zum Erreichen der optimalen Migration dar.

Die tatsächliche Größe der Transfermenge hängt von den verfügbaren Werten im Zustand eines Zustandssenders ab und von der ausgewählten Zustandstransferstrategie. Eine Zustandstransferstrategie beschreibt das lokale Transferverhalten, z. B. in welcher Reihenfolge Datentupel aus der Transfermenge ausgewählt und übertragen werden. Es wurden sechs Zustandstransferstrategien (OF, LF, RS, RSD, DS, SemS) und eine Erweiterung vorgestellt, deren Anwendung zu drei weiteren Zustandstransferstrategien (LF+,

RSD+, DS+) führt. Für alle wurden spezifische Formeln zur Ermittlung der endgültigen Transferparameter vorgestellt sowie Vor- und Nachteile diskutiert. Die tatsächliche Größe der Transfermenge wird anhand der ausgewählten Zustandstransferstrategie bestimmt und in der Zustandspaartabelle aktualisiert. Das globale Transferverhalten ist für den Transfer von mehreren Zuständen festzulegen. Es bestimmt u. a. die Ablaufreihenfolge der Zustände. Das Ergebnis der Festlegung von lokalem und globalem Transferverhalten ist eine aktualisierte, endgültige Zustandspaartabelle. Damit ist die Bestimmung der Konfigurationsparameter abgeschlossen.

Die Zustandspaartabelle wird zur Konfiguration der Migration genutzt. Entsprechend der Tabelle werden Operatoren und System mit temporären Funktionen erweitert. So erhalten Zustandssender eine Transferfunktion, die die Umsetzung der Zustandstransferstrategie realisiert. Zustandsempfänger werden u. a. mit zusätzlichen Transferwarteschlangen versehen, welche im Vergleich zu den Eingangswarteschlangen mit einer geringeren Priorität verarbeitet werden. Die Durchführung der Migration wird jeweils von einem Update-Manager pro Verarbeitungsknoten gemäß dem globalen Transferverhalten gesteuert.

Im Unterschied zu vergleichbaren Strategien ermöglicht der vorgestellte Ansatz, Zustände während des operativen Betriebs zu übertragen und trotzdem eine regelmäßige Ergebnisausgabe zu erhalten. Zudem ist eine Anwendung auf Anfragen in verteilten Umgebungen möglich. Durch die Verwendung der Zustandstransferstrategien und die Berechnung der Transfermenge werden nur notwendige Datentupel übertragen und eine optimale Migration ist möglich. Ein weiterer Vorteil der Zustandstransferstrategien ist die Flexibilität bei der Übertragungsreihenfolge, wodurch sich während der Migration genäherte Ergebnisse mit der Zielanfrage ermitteln lassen. Der Zustandstransfer ist für unbeteiligte Operatoren nicht wahrnehmbar, da er ausschließlich ungenutzte Systemressourcen verbraucht.

Von den vergleichbaren Migrationsstrategien ist lediglich GHM in der Lage, Datentupel parallel zum operativen Betrieb an die Zielanfrage zu übertragen. Die Ausgabe der Ergebnisse während der Migration wird durch eine priorisierte Originalanfrage gewährleistet. Der Transfer erfolgt, ähnlich OF, in der zeitlichen Reihenfolge, jedoch werden alle alten Werte der Zustände übertragen, nicht nur notwendige Werte. Deshalb ist für viele Anwendungsfälle die optimale Migration mit GHM nicht möglich. MS und GMS pausieren die Verarbeitung zugunsten des Zustandstransfers wodurch zeitweilig keine Ergebnisse produziert werden. Dies widerspricht in erster Linie der Anforderung, den operativen Betrieb nicht zu beeinflussen. PT, GPT und GenMig realisieren oder basieren auf dem Parallelprinzip, weshalb die Migrationsdauer der größten Anlaufzeit innerhalb der Anfrage entspricht. Durch den Verzicht auf den Zustandstransfer ist die Migrationsdauer von der optimalen Migrationsdauer weit entfernt. GenMig und HybMig sind außerdem für einen Transfer in verteilten Umgebungen ungeeignet, da z. B. Techniken wie „gemeinsame Zustände“ genutzt werden.

Kapitel 6

Anwendung auf aperiodische Datenströme

Ziel dieses Kapitels ist die Überführung der in Kapitel 5 vorgestellten Konzepte in eine geeignete Lösung für aperiodische Datenströme. Die Struktur des vorherigen Kapitels wird dabei im Wesentlichen beibehalten.

Ausgangspunkt für die Berechnung der Migrationsparameter ist auch in diesem Fall eine Tabelle von potentiellen Zustandspaaren für die Zustandsübertragung. Den Betrachtungen wird ein einfaches Modell eines aperiodischen Datenstroms zugrunde gelegt, welches kein Burst-Verhalten aufweist. Ein aperiodischer Datenstrom wird, wie in Abschnitt 2.2.1 eingeführt, durch einen mittleren Ereignisabstand $\overline{T}$ charakterisiert[1].

Zu den Erkenntnissen des vorherigen Kapitels zählt, dass eine Migration unter Zuhilfenahme eines Zustandstransfers schneller zu vollenden ist als mit dem Parallelprinzip. Ursache ist das Anlaufverhalten von zustandsbehafteten Operatoren. Eine Kombination beider Prinzipien wurde für die meisten Fälle als ideale Lösung identifiziert. Der Ablauf des Zustandstransfers ist in periodischen Systemen zwar vorhersagbar, für Anfragen mit mehreren Zuständen sind die Vorhersage und vor allem die Bestimmung der Migrationsparameter jedoch schwierig. Ein Verfahren für die Verwendung mit heterogenen Zuständen oder mit aperiodischen Datenströmen muss imstande sein, dieses Problem zu lösen. Außerdem wurde gezeigt, dass eine kürzere Migration möglich ist, wenn Zustandstransferstrategien verwendet werden, welche alte Werte zuerst übertragen und diese somit vor dem Verwerfen bewahren oder auch durch die Anwendung eines sofortigen Zustandstransfers. Dies funktioniert aber nur, wenn die Anzahl der zu übertragenden Werte exakt bestimmt werden kann. Aufgrund des aperiodischen Verhaltens ergeben sich deshalb neue Herausforderungen, die beim Entwickeln eines Verfahrens zur Bestimmung der Migrationsparameter beachtet werden müssen.

Alle charakteristischen Größen des Systems sind Näherungen, beispielsweise der mittlere Ereignisabstand. Aus diesem Grund lassen sich die Migrationsparameter nur abschätzen. Für jeden der zu transferierenden Zustände läuft die Abschätzung weiterhin auf die Anzahl von neuen und alten Werten im Zustand bei Migrationsende (N_n und N_o)

[1] Alle Parameter, die mittlere Werte repräsentieren und spezifisch für die aperiodischen Modelle sind, werden durch einen Überstrich gekennzeichnet, z. B. $\overline{T}$.

hinaus. Im Allgemeinen, d. h. bei $N_i < N_t$, treffen die neuen Werte in einem Zustand deutlich langsamer ein als die alten Werte, die durch Zustandstransfer hinzukommen. Bereits für einen einzelnen Operator resultieren daraus zwei Probleme. Wird N_o überschätzt und werden die Zustandswerte mit einer anderen Strategie als LF übertragen, dann dauert die Migration länger als nötig, da das Fenster bis zum Ende des Zustandstransfers mindestens eine Lücke aufweist. Wird andererseits N_o unterschätzt, dann endet der Zustandstransfer bevor das Fenster vollständig gefüllt ist und zur Vollendung der Migration muss auf zukünftige neue Werte gewartet werden. Es ist also ein Kompromiss zu finden, zwischen der Sicherheit, alle notwendigen Werte zu übertragen, und dem Bestreben, die Migration schnell zu beenden. Die Probleme sind auf Anfragen mit mehreren zustandsbehafteten Operatoren übertragbar.

Eine weitere Schwierigkeit entsteht durch die Durchführung des Zustandstransfers während der Pausenzeiten. Für periodische und schwankungsbeschränkte Datenströme konnte der Zustandstransfer rechtzeitig vor Eintreffen eines neuen Eingangswertes unterbrochen werden. Grundlage dafür war die Kenntnis der Pausendauer und die entsprechende Berechnung der Transferwerte pro Pause. Da das Verhalten eines aperiodischen Datenstroms nicht vorhersagbar ist, gilt dies auch für die Pausenzeiten. Deshalb kommt es zwangsläufig zu Kollisionen zwischen den Zustandstransferprozessen und der regulären Verarbeitung. Da eine Verzögerung den Anforderungen widerspricht, muss eine geeignete Kollisionsbehandlung erfolgen.

Nachfolgend wird ein einfaches heuristisches Verfahren für die Berechnung der Migrationsparameter beschrieben. Der Beschreibung der Heuristik wird die Betrachtung der funktionalen Erweiterungen vorangestellt, da diese wesentliche Grundlagen für die Berechnung behandelt.

6.1 Funktionale Erweiterungen

Zu den funktionalen Erweiterungen der Operatoren und des Systems (Abschnitt 5.4) sind zusätzliche Erweiterungen notwendig, um eine anforderungsgerechte Migration zu gewährleisten. Im Folgenden werden deshalb Erweiterungen für die Systemüberwachung, die Kollisionsbehandlung und die Migrationssteuerung erläutert.

Systemüberwachung Für die Berechnung der Migrationsparameter sind Systemparameter notwendig, beispielsweise der mittlere Ereignisabstand der Eingangsdatenströme ($\overline{T}$), die Verarbeitungszeiten (T_p) oder die Selektivität der Operatoren (sel). Neben der Möglichkeit, zuvor bestimmte statische Parameter zu verwenden, können die Systemparameter auch zur Laufzeit ermittelt werden. Dafür muss das Datenstromsystem die entsprechenden Systemelemente überwachen und die erfassten Daten zur Verfügung stellen. Manche DSMS, z. B. Aurora und Borealis [AAB+05], besitzen bereits eine solche Funktionalität. Außerdem können die inneren Zustände für die Bestimmung der Datenstromparameter genutzt werden, da sie eine bestimmte Anzahl vergangener Daten

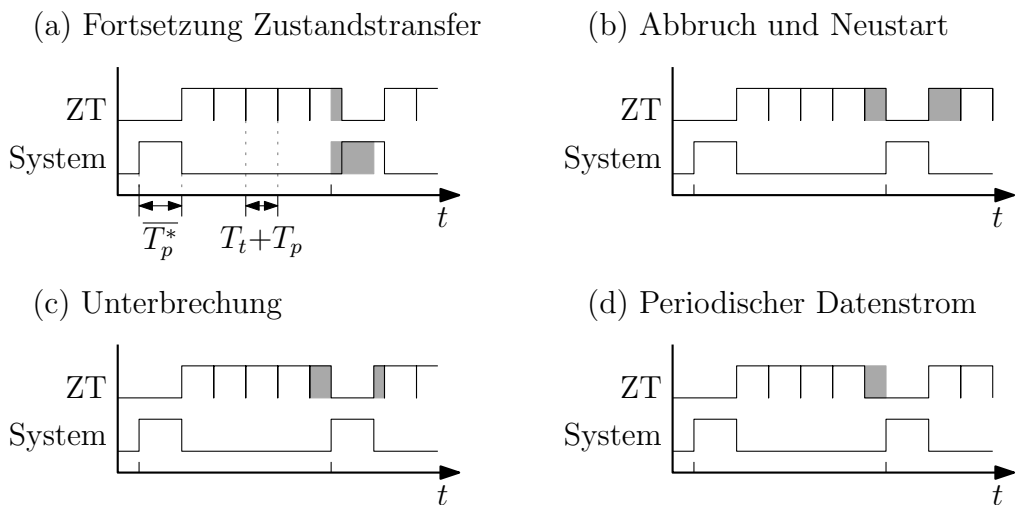


Abbildung 6.1: Verschiedene Szenarien der Kollisionsbehandlung, Besonderheiten durch graue Bereiche hervorgehoben; ZT – Zustandstransfer

enthalten. Aus den erfassten oder ermittelten Größen kann u. a. die Systemauslastung errechnet werden, d. h. der mittlere Ereignisabstand des gesamten Systems $\overline{T}_{syst}$ und die mittlere Verarbeitungsdauer des Systems $\overline{T}_p^*$. Alternativ ist eine direkte Beobachtung des Systems denkbar, vorausgesetzt die technischen Möglichkeiten erlauben dies.

Kollisionsbehandlung Eine der genannten Herausforderungen in aperiodischen Datenströmen ist der Umgang mit den unvermeidbaren Kollisionen zwischen Zustandstransfer und eingehenden Werten. Deshalb ist eine geeignete Kollisionsbehandlung notwendig, welche auch in der Berechnung der Migrationsparameter zu berücksichtigen ist. Die Kollisionsbehandlung beschränkt sich auf den aktuellen Übertragungswert bei Eintreffen eines neuen Eingangswertes. Generell lässt sich der Zustandstransfer dieses Wertes fortsetzen oder unterbrechen. Eine Fortsetzung der Übertragung des aktuellen Zustandswertes wird zu einer Verzögerung des operativen Betriebs führen. Eine Fortsetzung über den aktuellen Zustandstransferwert hinaus würde die Verarbeitung zusätzlich verzögern, weshalb diese Möglichkeit ausgeschlossen wird.

In Abbildung 6.1 sind drei verschiedene Methoden der Kollisionsbehandlung dargestellt. Diese werden ergänzt durch das Verhalten eines periodischen Datenstroms. Für jedes Szenario ist durch ein Impulsdiagramm die Auslastung des Systems durch die reguläre Verarbeitung von Eingangswerten (System) sowie die Auslastung durch den Zustandstransfer (ZT) angegeben. Die tatsächliche Ereigniszeit der Eingangswerte ist jeweils durch eine Markierung auf der Abszisse angegeben. Die Impulsdauer steht entweder für die Verarbeitungszeit $\overline{T}_p^*$ (System) oder für die Summe aus Tupeltransferzeit T_t und eventueller Verarbeitungszeit T_p für einen Zustandstransferwert (ZT). Die Besonderheiten jedes Szenarios sind durch graue Bereiche hervorgehoben.

Die Fortsetzung des Zustandstransfers zu Lasten der regulären Verarbeitung wird in Abbildung 6.1a gezeigt. Bei Eintreffen eines neuen Wertes bei der Datenquelle wird zunächst der Transfer des aktuellen Zustandswertes vollendet. Anschließend wird der neue Eingangswert verarbeitet. Dessen eigentliche Verarbeitungszeit, d. h. ohne Zustandstransfer, repräsentiert die graue Box (System). Der Zustandstransfer wird danach mit dem nächsten Zustandswert fortgesetzt. Die Zeit zum Vollenden der Übertragung des Zustandswertes (grau, ZT) entspricht der Verzögerung der regulären Verarbeitung. Die Fortsetzung des Transfers ist besonders bei großen T_t kritisch, da dies zu erheblichen Verzögerungen führen kann. Daraus folgt für dieses Szenario, je größer T_t, desto größer ist die durchschnittliche Verzögerung. Sofern T_t konstant ist und die Wahrscheinlichkeit einer Unterbrechung jederzeit gleich groß ist, beträgt die durchschnittliche Verzögerung $T_t/2$ und die maximale Verzögerung T_t. Für variable T_t ist die maximale Verzögerung $\max(\{T_t\})$. Diese Kollisionsbehandlungsmethode kann in Betracht gezogen werden, falls die maximale Verzögerung unterhalb der geforderten Maximalverzögerung liegt.

Abbildung 6.1b veranschaulicht den Abbruch des Zustandstransfers zu Gunsten der regulären Verarbeitung mit anschließendem Neustart des letzten Übertragungswertes. Die grauen Boxen repräsentieren den gleichen Zustandswert bei zwei Übertragungsversuchen. Erst der zweite Versuch ist erfolgreich. Der Vorteil dieses Szenarios gegenüber dem vorherigen ist, dass der reguläre Prozess nicht verzögert wird. Nachteilig wirkt sich jedoch der Abbruch aus, da dadurch Transferzeit nutzlos verbraucht wird, insbesondere bei großen T_t.

Das dritte Szenario, dargestellt in Abbildung 6.1c, unterbricht den Zustandstransfer zu Gunsten des operativen Betriebs. Auch hier markiert die graue Box den gleichen Wert, dessen Übertragung wird aber nach der Verarbeitung des Eingangswertes fortgesetzt. Dadurch wird keine Transferzeit nutzlos verbraucht und der operative Betrieb wird nicht verzögert. Im Prinzip stellt dies die ideale Lösung dar. Es erfordert jedoch die Funktionalität, die Übertragung von Werten unterbrechen und fortsetzen zu können. Inwiefern diese Funktionalität realisierbar ist, bleibt zu überprüfen.

Als Vergleich beinhaltet Abbildung 6.1d die Realisierung in periodischen oder schwankungsbeschränkten Datenströmen. Aufgrund der Kenntnis der Periodendauer und der Verarbeitungszeit, kann für die verbleibende Pausenzeit eine Anzahl von Zustandstransferwerten (N_t), wie in Kapitel 5 beschrieben, errechnet werden. Die graue Box symbolisiert die Zeit, welche in Erwartung des nächsten Eingangswertes nicht für den Zustandstransfer genutzt wird.

Die vorgestellten Szenarien lassen sich in Bezug auf die Berechnung der Migrationsparameter in zwei Gruppen unterteilen. Die erste Gruppe umfasst die Szenarien (a) und (c), weil diese einen verlustfreien Zustandstransfer ermöglichen. Der anderen Gruppe gehört Szenario (b) an, da die Zustandstransferzeit nicht vollständig ausgenutzt werden kann. Welche Kollisionsbehandlung für ein System gewählt wird, hängt maßgeblich von den Funktionen des Systems ab. Es ist durchaus möglich, dass aufgrund der Implementierung eines Systems manche Kollisionsbehandlungsmethoden nicht unterstützt werden können. Durch die Einführung von getrennten Warteschlangen (Abschnitt 5.4.1) für die

Verarbeitung und den Zustandstransfer ist zumindest eine funktionale Grundlage für eine Priorisierung der Prozesse gegeben.

Migrationssteuerung Mit dem numerischen Modell für periodische und schwankungsbeschränkte Datenströme kann mit N_o die exakte Anzahl von Zustandstransferwerten berechnet werden. In aperiodischen Datenströmen handelt es sich nur um eine Näherung. Es kann deshalb zu Situationen kommen, die in periodischen Datenströmen nicht auftreten. Überträgt man beispielsweise einen Zustand mit OF in ein anzahlbasiertes Fenster, und der mittlere Ereignisabstand während der Migration ist geringer als der für die Berechnung verwendete, dann wird ein vollständig gefülltes Fenster gegebenenfalls eher erreicht. Dies kann aber auch bedeuten, dass noch nicht alle alten Werte im Fenster enthalten sind. Genau in diesem Fall existiert eine Lücke zwischen den alten und den neuen Werten und der Zustandstransfer ist fortzusetzen, bis diese Lücke geschlossen ist. Die Endbedingung der Migration kann also unter Umständen nicht nur vom Erreichen der neuen Fenstergröße abhängig gemacht werden, sondern muss zusätzliche Kriterien beinhalten. Dies ist bei der Migrationssteuerung zu berücksichtigen, z. B. im Transfermanager.

6.2 Zustandsauswahl

Die Anwendung der Konzepte des vorherigen Kapitels war auf Systeme mit einem periodischen Systemverhalten beschränkt, was nur durch spezielle Konstellationen von Eingangsdatenströmen (Varianten (a) und (b), Abbildung 5.1) möglich ist. Die nachfolgenden Konzepte sind prinzipiell für jedes Systemverhalten verwendbar, da zum einen das Systemverhalten mit Näherungen abgebildet wird, und zum anderen durch Kollisionsbehandlung die notwendigen funktionalen Grundlagen gegeben sind, um eine Migration mit Zustandstransfer durchzuführen.

Sind die Systemeigenschaften bekannt, d. h. der mittlere Ereignisabstand $\overline{T}_{syst}$ und die mittlere Verarbeitungszeit $\overline{T}_p^*$, kann daraus die mittlere Pausendauer des Systems $\overline{T}_i$ berechnet werden.

$$\overline{T}_i = \overline{T}_{syst} - \overline{T}_p^* \tag{6.1}$$

Nachfolgend wird wie in Kapitel 5 die Berechnung für Anfragen mit einem einzelnen Zustand und für Anfragen mit mehreren Zuständen gezeigt. Die Zustände repräsentieren jeweils gleitende Fenster. Die vorgestellten Konzepte bauen teilweise auf den Formeln des vorherigen Kapitels auf, was entsprechend erwähnt wird. Die Kollisionsbehandlung wird bei der Berechnung berücksichtigt. Dabei werden Szenario (a) stellvertretend für die erste Gruppe und Szenario (b) für die zweite Gruppe verwendet.

6.2.1 Berechnung für Anfragen mit einem Zustand

Der Ablauf zur Berechnung der Migrationsparameter gleicht prinzipiell dem Verfahren für periodische und schwankungsbeschränkte Datenströme. Basierend auf den System- und Anfrageeigenschaften werden Anteile der neuen und alten Werte im Zustand bei Migrationsende ermittelt. Da es sich bei der Berechnung um eine Abschätzung handelt, können einige Vereinfachungen getroffen werden. Außerdem entfallen manche Funktionen, wie z. B. sofortiger Zustandstransfer, da sie für dieses Modell nicht relevant sind. Die Berechnung wird wieder anzahlbasiert durchgeführt, weshalb eine Umrechnung von zeitbasierten Fenstergrößen notwendig ist. Unter Verwendung des mittleren Ereignisabstands des Eingangsdatenstroms kann dabei wieder Gleichung (5.11) verwendet werden.

Gemäß der Kollisionsbehandlung ist zunächst die Größe N_t zu berechnen. Diese gibt an, wie viele Tupel während einer durchschnittlichen Pause ($\overline{T}_i$) per Zustandstransfer zum neuen Zustand übertragen werden können. Dabei sind deren Tupeltransferzeit (T_t) und Verarbeitungszeit (T_p) zu berücksichtigen. Beide Größen können, wie bei den periodischen Strömen, spezifisch für einen Tupeltyp sein oder als mittlere Werte verwendet werden, falls sich die Werte pro Tupel unterscheiden sollten. Die Berechnung erfolgt ähnlich der Gleichung (5.10). Es wird jedoch auf die Berücksichtigung einer Reserve verzichtet, d. h. $r = 0$. Die resultierende Formel für die jeweilige Gruppe befindet sich in den Gleichungen (6.2). Für die zweite Gruppe wird die Abrundungsfunktion benutzt, da jeweils der Transfer des kollidierenden Wertes abgebrochen und später neu gestartet wird. Da dies für die erste Gruppe nicht zutrifft, wird dort auf die Abrundungsfunktion verzichtet.

Szenario (a)	Szenario (b)
$N_t = \dfrac{\overline{T}_i}{T_t + T_p}$	$N_t = \left\lfloor \dfrac{\overline{T}_i}{T_t + T_p} \right\rfloor$

(6.2)

Anschließend kann aus N_t die Länge des Zustandstransfers abgeschätzt werden. Da sich die Berechnung allerdings auf das Systemverhalten ($\overline{T}_i$) und nicht nur auf den Eingangsdatenstrom bezieht, ist N_i zuvor im Verhältnis zum Gesamtsystem zu bestimmen.

$$\overline{N}_i = \frac{N_i}{\overline{T}} * \overline{T}_{syst} \tag{6.3}$$

Mit $N_t + \overline{N}_i$ ist somit die „Anzahl“ von Tupeln bekannt, die pro Systemzyklus der Länge $\overline{T}_{syst}$ in das neue Fenster gelangen können. Daraus ergibt sich N_c (Gleichung (6.4)), ähnlich der Gleichung (5.12). Zwar gibt es im aperiodischen System faktisch keine Zyklen, aber für die Abschätzung gibt dieser Wert wieder, wie lange die Migration dauert, gemessen am mittleren Ereignisabstand des Systems.

$$N_c = \frac{w_n}{N_t + \overline{N}_i} \tag{6.4}$$

Schließlich kann daraus die Anzahl der neuen Werte im Fenster bei Migrationsende berechnet werden. In Gleichung (6.5) ist die entsprechende Formel wiedergegeben. Diese ist eine Vereinfachung von Gleichung (5.15), da der Transfer im Unterschied zu den periodischen Datenströmen nicht notwendigerweise mit neuen Werten beginnt.

$$N_n = \left\lfloor \overline{N}_i * N_c \right\rfloor \tag{6.5}$$

$$T_M = \frac{N_n}{N_i} * \overline{T} \tag{6.6}$$

Die Anzahl der alten Werte zur Vervollständigung des Fensters berechnet sich wie in Gleichung (5.16) angegeben. Damit sind N_o und N_n für den Zustand bekannt und auch die Migrationsdauer kann unter Verwendung von N_n abgeschätzt werden (Gleichung (6.6)).

Somit existiert eine einfache Abschätzung für die Ermittlung der Migrationsparameter für einen Zustand. Die Unterschiede bei der Kollisionsbehandlung wirken sich lediglich auf die Berechnung von N_t aus. Alle weiteren Berechnungen sind davon unabhängig.

6.2.2 Berechnung für Anfragen mit mehreren Zuständen

Die Betrachtungen in Kapitel 5 haben gezeigt, dass insbesondere bei der Migration mit mehreren Zuständen viele Schwierigkeiten bei der Bestimmung der Migrationsparameter zu erwarten sind und diese oft nur durch Einzelfallbetrachtung ermittelt werden können. Insbesondere heterogene Zustände und Abhängigkeiten zwischen Zuständen tragen dazu bei. Deshalb beschränkten sich die vorgestellten Lösungen auf homogene Zustände. Nachfolgend wird eine einfache Heuristik vorgestellt, welche die Bestimmung der Migrationsparameter durch ein Näherungsverfahren ermöglicht und sich auch auf heterogene und abhängige Zustände anwenden lässt. Die Heuristik erhebt keinen Anspruch auf die Ermittlung der optimalen Migrationsparameter.

Der Ablauf umfasst im Wesentlichen drei Phasen. In der ersten Phase werden die relevanten Zustände[2] nach ihrer zeitlichen Größe geordnet. Dafür ist es notwendig, für anzahlbasierte Fenster eine entsprechende Zeit w_n^* zu ermitteln, z. B. durch Berechnung basierend auf den Eigenschaften des Eingangsdatenstroms ($w_n^* \approx (w_n/N_i)*\overline{T}$) oder durch Untersuchung der Werte, die aktuell im Fenster gespeichert sind. Die Durchführung der Parameterbestimmung wird nachfolgend am Beispiel von vier heterogenen Zuständen $\mathcal{Z}_A$, $\mathcal{Z}_B$, $\mathcal{Z}_C$ und $\mathcal{Z}_D$ mit $w_B^* > w_C^* > w_A^* > w_D^*$ demonstriert. Ihre Sortierung ist in Abbildung 6.2 dargestellt.

In der nächsten Phase wird schrittweise, für jeden Zustand und beginnend mit dem größten, ein Bedarf an Migrationszeit berechnet. Die einzelnen Berechnungsschritte sind hypothetisch. Das Ergebnis dieser Phase ist die Abschätzung der gesamten Migrationsdauer (T_M), mit der Absicht, dass alle Zustände bei Migrationsende nahezu gleichzeitig

[2] beispielsweise aus der Zustandspaartabelle

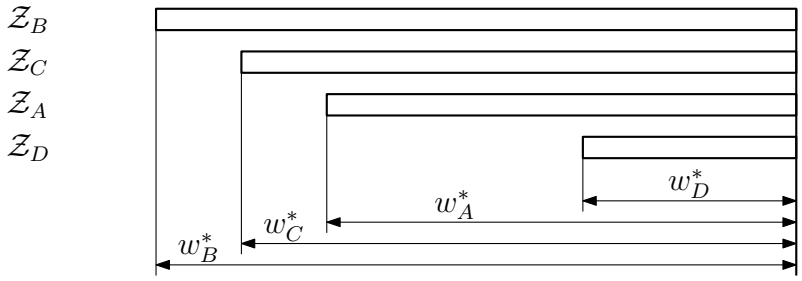


Abbildung 6.2: Fenstergrößen der beteiligten Zustände

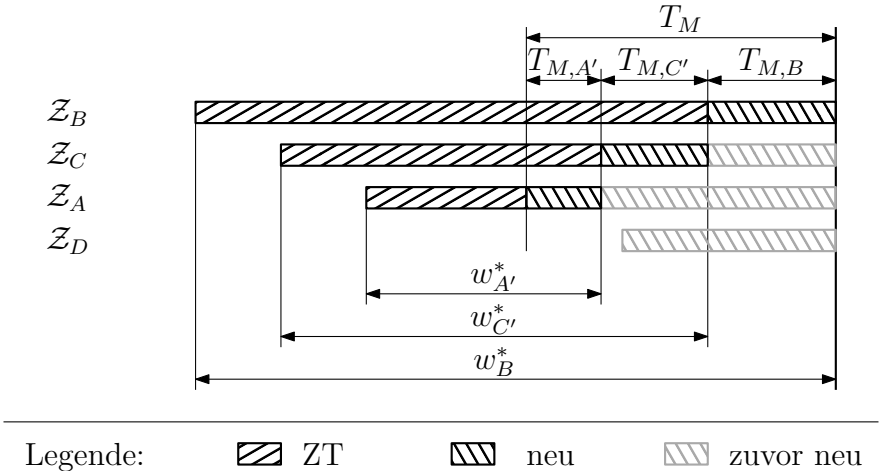


Abbildung 6.3: Bestimmung von T_M

vollständig sind. Im Beispiel (Abbildung 6.3) wird mit $\mathcal{Z}_B$ begonnen, da dessen zeitliche Fenstergröße die größte ist. Mit der im vorherigen Abschnitt beschriebenen Vorgehensweise für einzelne Zustände wird die Migrationsdauer für diesen Zustand ($T_{M,B}$) abgeschätzt. Nächster Zustand bei der Berechnung ist $\mathcal{Z}_C$, allerdings nur mit einem Teil seines Fensters $w^*_{C'} = w^*_C - T_{M,B}$. Grundüberlegung dieser Reduktion ist, dass während der Migration von $\mathcal{Z}_B$ auch die anderen Zustände mit neuen Werten versorgt werden. Es ist also nur das verbleibende Fenster per Migration mit neuen und alten Werten zu füllen. Daraus ergibt sich $T_{M,C'}$. Bereits zu diesem Zeitpunkt ist zu erkennen, dass der Zustand $\mathcal{Z}_D$ während der Migration vollständig durch neue Werte gefüllt werden kann. Deshalb ist für ihn kein Zustandstransfer notwendig. Zuletzt wird $T_{M,A'}$ für $\mathcal{Z}_A$ mit $w^*_{A'} = w^*_A - T_{M,B} - T_{M,C'}$ ermittelt. Die geschätzte, gesamte Migrationsdauer ist schließlich die Summe aller einzelnen Anteile, d. h. $T_M = T_{M,B} + T_{M,C'} + T_{M,A'}$.

In der letzten Phase (Abbildung 6.4) werden basierend auf T_M die tatsächlichen Migrationsparameter bestimmt. So kann für die Zustände $\mathcal{Z}_B$, $\mathcal{Z}_C$ und $\mathcal{Z}_A$ die Anzahl neuer Werte direkt aus der geschätzten Migrationsdauer und den Eigenschaften des Eingangsdatenstroms berechnet werden.

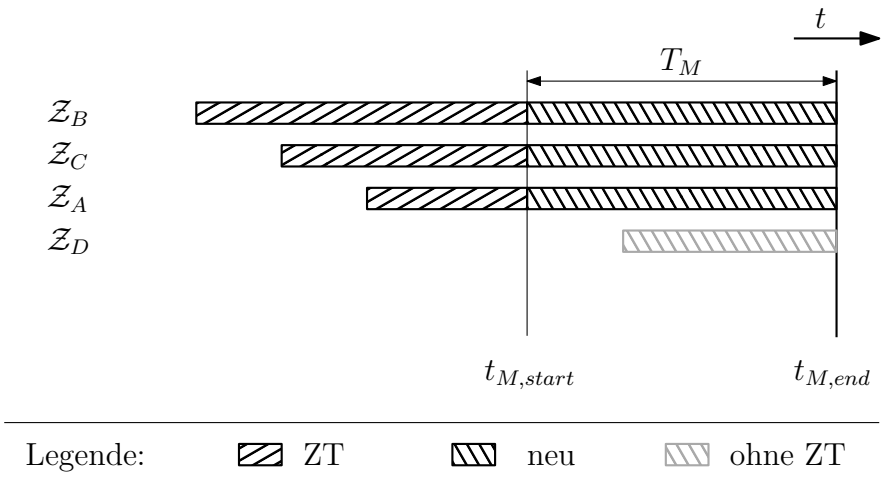


Abbildung 6.4: Bestimmung der tatsächlichen Migrationsparameter

$$N_n = \left\lfloor \frac{T_M}{\overline{T}} * N_i \right\rfloor \tag{6.7}$$

Die Größe N_o ergibt sich wie zuvor jeweils aus der Differenz zwischen geplanter Fenstergröße und N_n (Gleichung (5.16)). Wie erläutert, kann der Zustand $\mathcal{Z}_D$ für die Migrationssteuerung ignoriert werden, da die Migrationsdauer seine Fenstergröße übersteigt und deshalb kein Zustandstransfer notwendig ist.

Mit dem vorgestellten heuristischen Verfahren ist es möglich, die Transferparameter für Szenarien mit mehreren Zuständen abzuschätzen. Dabei können sowohl heterogene als auch abhängige Zustände in der Zustandspaarmenge enthalten sein. Durch die Verwendung von zustandsspezifischen Parametern, z. B. $\overline{N}_{i,B}$, kann das Verhalten ihrer Eingangsdatenströme separat geschätzt werden und eine Berechnung der Migrationsdauer wird möglich. Das Problem der einfachen, sequentiellen Abschätzung ist, dass die Migrationsdauer und damit N_n überschätzt werden. So existiert beispielsweise in $T_{M,B}$ ein Anteil für den Transfer von Werten, welche letztendlich während $T_{M,A'}$ und $T_{M,C'}$ als neue Werte in den Zustand gelangen. Ein Zustandstransfer ist für diese Werte also nicht notwendig und $T_{M,B}$ wäre geringer. Dieser Effekt tritt auch bei $\mathcal{Z}_C$ auf. Durch die Einführung einer zusätzlichen Phase zur iterativen Kompensation oder durch die Wahl einer geeigneten Zustandstransferstrategie kann dieser Effekt vermindert bzw. vernachlässigt werden.

6.3 Zustandstransferstrategien

Mit der Auswahl der Zustandstransferstrategien ist das lokale Transferverhalten und anschließend das globale Transferverhalten festzulegen. Die Besonderheiten im Zusammenhang mit aperiodischen Datenströmen werden an dieser Stelle kurz skizziert.

Lokales Transferverhalten Für die Festlegung des lokalen Transferverhaltens stehen die unter Abschnitt 5.3.1 beschriebenen Zustandstransferstrategien zur Verfügung. Für jeden Zustand kann die Zustandstransferstrategie individuell festgelegt werden. Bei periodischen Datenströmen war die wesentliche Eigenschaft zum Erreichen einer optimalen Migration die Anzahl der verfügbaren Zustandswerte im Originalzustand. Dafür wurde für jede Zustandstransferstrategie die Größe $w_{o,*}$ bestimmt, welche die minimale Fenstergröße des Originalzustands zum Erreichen einer optimalen Migration angibt. Prinzipiell lässt sich diese Größe auch für ein System mit aperiodischen Datenströmen berechnen. Es ist aber zu bedenken, dass es sich nur um eine ungefähre Richtgröße handelt. Aufgrund des veränderlichen Verhaltens der Datenströme kann keine genaue Aussage darüber getroffen werden.

Außerdem hat die Übertragungsreihenfolge der Zustandswerte verschiedene Konsequenzen. Die Problematik von OF wurde bereits in der Kapiteleinleitung beschrieben. Wird N_o unterschätzt, werden nicht genügend Werte übertragen und die Migration verlängert sich durch das Vervollständigen des Fensters mit neuen Werten. Wird N_o überschätzt, kann es zu einer Lücke kommen, welche den Zustandstransfer und dadurch die Migration verlängert. Zur Vermeidung der Lücke kann LF benutzt werden. Dies kann aber dazu führen, dass alte Werte im Originalzustand verworfen werden, bevor sie zum neuen übertragen werden konnten. Die Verwendung von LF+ bedarf, wie jede andere Strategie, die auf die Kenntnis von N_o angewiesen ist, einer sehr guten Abschätzung von N_o.

Globales Transferverhalten Das globale Transferverhalten war für die periodischen Datenströme hauptsächlich durch die Reihenfolge der Zustände und die Gruppengröße der Transfertupel der einzelnen Zustände bestimmt. Diese Größen können für aperiodische Migrationsszenarien, sofern relevant, auch angegeben werden. Die Gruppengröße kann aus den N_o der Zustände abgeleitet werden. Das resultierende Größenverhältnis sorgt für eine gleichmäßige Zustandsübertragung und ein nahezu gleichzeitiges Ende des Zustandstransfers für alle Zustände. Alle Parameter aus der Festlegung des lokalen und globalen Transferverhaltens werden in der Zustandspaartabelle (ähnlich Tabelle 5.3) gespeichert und stehen somit für die Durchführung der Migration zur Verfügung.

In Migrationsszenarien mit unterschiedlich großen Tupeltransferzeiten (T_t) kann gegebenenfalls die Übertragungsreihenfolge der Zustandswerte einen Ansatzpunkt für eine Optimierung bieten. Besteht eine Möglichkeit, die Ereigniszeit der Eingangsdaten mit einer gewissen Genauigkeit vorherzusagen, z. B. in schwankungsbeschränkten Datenströmen, dann ist es für die Szenarien (a) und (b) hilfreich während einer Pause die Werte mit großen T_t zuerst zu übertragen und die mit kleinen T_t zum Ende der Pause. Für Szenario (a) resultiert daraus eine geringere Verzögerung der regulären Verarbeitung. Bei Szenario (b) ist der Verlust von Transferzeit durch den Abbruch kleiner. Diese Erweiterung ist allerdings nicht ausschließlich mittels Zustandspaartabelle realisierbar und erfordert demgemäß eine andere Art der Konfiguration.

6.4 Erweiterungsmöglichkeiten

Das heuristische Verfahren stellt ein einfaches Konzept zur Abschätzung der Migrationsdauer dar und bietet Möglichkeiten zur Erweiterung, um die Vorhersage zu verbessern, die Optimierung der Migration zu berücksichtigen oder robust auf veränderliche Systemeigenschaften zu reagieren. Einige Richtungen für mögliche Weiterentwicklungen werden im Folgenden aufgezeigt.

- *Systemmodell:* Die Vorhersage basiert auf einem sehr abstrakten Systemmodell mit mittleren Ereignisabständen für Datenströme, mittleren Verarbeitungszeiten für Datenstromwerte und mittleren Tupeltransferzeiten. Ein komplexeres Systemmodell könnte zu einer genaueren Vorhersage beitragen, da das wahre Systemverhalten besser abgebildet wird, beispielsweise Burst-Verhalten.

- *Fenstertypen:* Die Heuristik funktioniert in der dargestellten Form für gleitende und für wachsende Fenster. Für springende Fenster kann sie zwar auch benutzt werden, es ist jedoch keine optimale Migration zu erwarten. Ursache ist, dass das Springen des Fensters in der Heuristik nicht berücksichtigt wird. Dafür wäre die Kenntnis der aktuellen Zeit und der nächsten Sprungzeit notwendig. Eine Migration mit Zustandstransfer ist dann evtl. nicht notwendig. Ebenso ist die Anwendbarkeit in Szenarien mit anderen Fenstertypen zu untersuchen.

- *Laufzeitanpassungen:* Da sich die Eigenschaften besonders in aperiodischen Datenströmen fortlaufend ändern können, kann es hilfreich sein, auch die Migrationsparameter kontinuierlich neu zu berechnen und gegebenenfalls anzupassen. Dafür ist eine fortlaufende Überwachung des Systems unabdingbar. Eine dynamische Anpassung zur Laufzeit ist insbesondere für Systeme mit Burst-Verhalten interessant.

- *Sicherheiten:* Das Modell für die periodischen Systeme verfügt mit der optionalen Reserve r über eine Stellgröße zur Vermeidung von Kollisionen. Zwar ist für aperiodische Systeme eine solche Größe nicht notwendig, da sich Kollisionen ohnehin nicht vermeiden lassen, stattdessen treten andere Schwierigkeiten auf, die mit einer ähnlichen Sicherheitsgröße kompensiert oder abgeschwächt werden können. Vor allem die Abschätzung von N_o erfordert einen Kompromiss, um hinreichend viele Werte zu übertragen, die Migration aber nicht unnötig zu verlängern. Mit Hilfe eines geeigneten Sicherheitsfaktors kann das Risiko, N_o zu unterschätzen, reduziert werden. In Kombination mit Laufzeitanpassungen könnte dies eine vielversprechende Lösungsmöglichkeit zur Beherrschung dieser Problematik darstellen.

- *Heuristik:* Die Heuristik basiert auf der Sortierung der Zustände nach ihrer zeitlichen Größe des neuen Fensters (erste Phase). Abhängig vom Szenario können unter Umständen auch andere Sortierungen interessant sein, beispielsweise wie viele Werte in w_o verfügbar sind oder wie schnell Werte übertragen werden können. Eine Berücksichtigung von anderen Eigenschaften oder die Verwendung von

zusätzlichen Kriterien könnten die Abschätzung verbessern. Eine Beschränkung der Heuristik ist, dass die berechneten Migrationsparameter nur dann zutreffen, wenn hinreichend viele Werte in den Originalzuständen vorhanden sind. Wird beispielsweise eine Migration benutzt, um eine Vergrößerung eines anzahlbasierten Fensters durchzuführen, verlängert sich die Migrationsdauer durch fehlende Werte zur Vervollständigung des Fensters. Dies wird zwar bei der Berechnung der einzelnen Zustände berücksichtigt, kann sich aber auf alle anderen Migrationszustände auswirken, da deren Zustandstransfer gegebenenfalls reduziert werden kann. Dadurch wird wiederum Transferkapazität für das vergrößerte Fenster frei. Mit der Weiterentwicklung der Heuristik zu einem iterativen Verfahren kann eine genauere Abschätzung erreicht werden.

- *Zustandstransferstrategien:* Die Verwendung der Zustandstransferstrategie SemS wurde im Zusammenhang mit aperiodischen Datenströmen nicht betrachtet. Aufgrund ihrer Eigenschaft, eine schnelle Umschaltung zur neuen Anfrage zu ermöglichen, sollte eine Anwendung in aperiodischen Systemen und der Einfluss auf die Ermittlung der Migrationsparameter untersucht werden. Wegen der vielfältigen Eigenschaften und Herausforderungen von aperiodischen Datenströmen erscheint auch die Entwicklung neuer, spezieller Zustandstransferstrategien möglich.

6.5 Zusammenfassung

Dieses Kapitel beschreibt ein allgemeines, heuristisches Näherungsverfahren für die Bestimmung von Migrationsparametern in aperiodischen Datenströmen. Während die Konzepte im vorherigen Kapitel mit vertretbarem Aufwand nur auf homogene Zustände in periodischen Datenströmen anwendbar sind, kann das heuristische Verfahren auch für heterogene und abhängige Zustände sowie für periodische und schwankungsbeschränkte Datenströme verwendet werden. Die Heuristik kann als Grundkonzept für optimierte Näherungsverfahren genutzt werden. Einige Erweiterungsmöglichkeiten wurden angedeutet.

Kapitel 7

Evaluierung

In diesem Kapitel werden die Einzelheiten zur Implementierung und Evaluierung der entwickelten Transferstrategien beschrieben. Es werden zunächst die Details der für die Evaluierung verwendeten technischen Prototypen erläutert. Daran schließen sich Bemerkungen zu deren Laufzeiteigenschaften und zur Bestimmung der Korrektheit der Transfermethoden an. Nach einer Funktionsanalyse der entwickelten Transferstrategien wird ihre Performance anhand von Beispielen aus dem Umfeld der zustandserhaltenden Wartung untersucht und mit den existierenden Migrationsverfahren verglichen.

7.1 Prototypische Implementierung

Die neuen Migrationsstrategien und einige der existierenden Verfahren wurden sowohl mit einem OSGi-basierten Prototyp als auch simulativ analysiert. Die Komponenten des Prototyps, seine Funktionsweise und seine Eigenschaften in Bezug auf die Migrationsexperimente sind nachfolgend beschrieben. Details zur Verwendung der Simulationsskripte werden in den Abschnitten über die Evaluierung der Transferstrategien im hinteren Teil des Kapitels erläutert.

7.1.1 OSGi-basierter Prototyp

OSGi [OSGi09, OSGi11] ist eine modulare Java-Middleware, welche es erlaubt einzelne Module, sogenannte Bundles, während der Laufzeit zu starten und zu stoppen. Dadurch ist es möglich, Bundles zu aktualisieren, indem eine Version mit verändertem Quellcode geladen, kompiliert und ausgeführt wird. Vor allem deshalb wurde OSGi als geeignete Technologie für den DSMS-Prototyp identifiziert, dessen Verarbeitungslogik sich dadurch zur Laufzeit uneingeschränkt verändern lässt. In Abbildung 7.1 ist die Softwarearchitektur des Prototyps dargestellt.

Jede Komponente repräsentiert ein OSGi-Bundle. Der DS-Manager, der Algorithmen-Pool und die Systemkomponenten-Bibliothek stellen die grundlegenden Funktionen des DSMS zur Verfügung. Alle anderen Bundles sind optional und werden zur Unterscheidung mit einem gestrichelten Rahmen abgebildet. Die Aufgaben der einzelnen Bundles werden nachfolgend im Einzelnen erklärt.

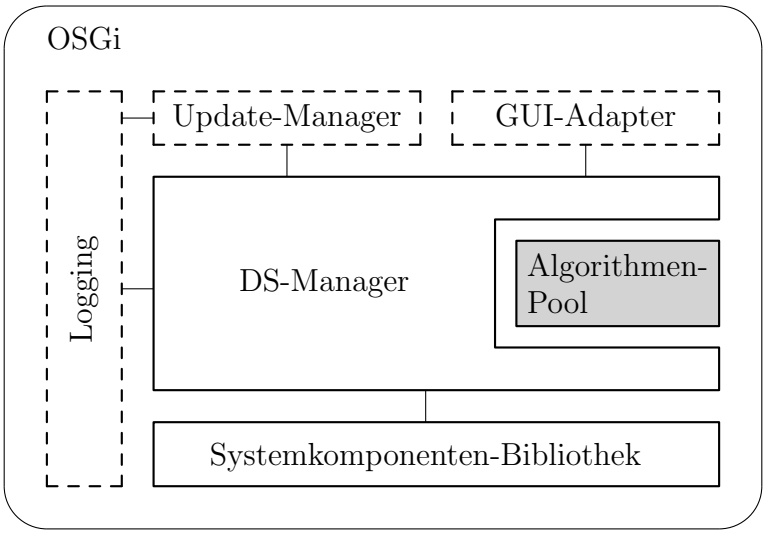


Abbildung 7.1: Softwarearchitektur des Prototyps

Systemkomponenten-Bibliothek Die Systemkomponenten-Bibliothek beinhaltet alle Basiselemente für das DSMS, darunter sowohl die abstrakten Klassen, z. B. der Systemelemente oder der Warteschlangen, als auch deren konkrete Implementierungen. Außerdem sind die Schnittstellen (Interfaces) enthalten, die von den Klassen implementiert werden, sowie die Formatdefinitionen für den Datenaustausch zwischen den Systemelementen.

Für die Kommunikation im System wird das OSGi-spezifische Entwurfsmuster „Whiteboard Pattern“ [KH04] verwendet. Dieses ist eine Variante des Beobachter-Entwurfsmusters (auch „Observer“ [GoF94] oder „Listener“) zur Realisierung einer Publish-Subscribe-Kommunikation. Die Besonderheit ist, dass für das Registrieren die OSGi-Service-Registry genutzt wird. Dafür melden sich die Datenempfänger unter Angabe einer DSID bei der Service-Registry an. Die DSID bezieht sich dabei auf einen Ausgangsdatenstrom eines Operators, d. h. der Datensender wird mittels DSID benannt. Ein Datensender kann die Service-Registry nach interessierten Empfängern abfragen und sendet seine Ergebnisse an die entsprechenden Systemelemente, indem sie in die dazugehörigen Warteschlangen eingefügt werden. Es können sich mehrere Empfänger für die gleiche DSID registrieren. Diese Kommunikation wird zwischen Systemelementen für die Datenübertragung und für den Zustandstransfer gleichermaßen genutzt.

DS-Manager Der DS-Manager hat im Wesentlichen zwei Aufgaben. Er erzeugt, aktualisiert und entfernt Systemelemente und steuert den Ablauf der Datenstromverarbeitung. Darüber hinaus führt er vor und während der Migration verschiedene Aktionen auf Anweisung des Update-Managers aus.

Basierend auf den Klassen der Systemkomponenten-Bibliothek werden die Systemele-

mente durch den DS-Manager instanziiert. Dafür benötigt der DS-Manager außerdem eine Anfragenbeschreibung, d. h. welche Systemelemente sind in der Anfrage enthalten und wie sind diese miteinander verbunden. Für Datenquellen und Datensenken werden die Verbindungen zu den externen Systemen aufgebaut. Zudem besteht die Möglichkeit, Datenquellen zu simulieren. Dafür werden Parameter wie der Datenstromtyp oder der (mittlere) Ereignisabstand sowie ein Algorithmus zum Generieren von Datentupeln festgelegt. Die Datentupel können dabei je nach Anfragetyp verschieden gestaltet sein, z. B. Fließkomma- oder Integer-Werte für Aggregationsoperatoren oder komplexe Tupel für Verbundoperatoren. Das OSGi-System sorgt dann zur Laufzeit für die Erzeugung der Datenstromelemente gemäß der Konfiguration.

Beim Erzeugen einer Operatorinstanz ist ein Algorithmus anzugeben, welcher referenziert wird und im laufenden Betrieb die Eingangsdaten zu Ergebnissen verarbeitet. Dabei stehen alle Algorithmen des Algorithmen-Pools zur Verfügung. Die Verbindung zwischen den Systemelementen erfolgt, wie zuvor beschrieben, mittels Publish-Subscribe-Kommunikation unter Angabe der entsprechenden DSID.

In Vorbereitung einer Migration kann der Update-Manager den DS-Manager anweisen, die beteiligten Systemelemente gemäß der Migrationskonfiguration zu erweitern. Der DS-Manager erstellt dann Referenzen zu Zustandstransferstrategien, welche ebenfalls im Algorithmen-Pool enthalten sind, erzeugt zusätzliche Warteschlangen für den Zustandstransfer und stellt schließlich die Verbindung zwischen Zustandsempfängern und Zustandssendern her.

Die Ablaufsteuerung der Datenstromverarbeitung durch den DS-Manager bezieht sich vorrangig auf die Ausführungsreihenfolge von Systemelementen und die Verarbeitungsreihenfolge von Eingangswerten im gesamten System. Üblicherweise werden diese Aufgaben durch Scheduler realisiert. Deren Optimierung ist einer der Forschungsschwerpunkte in existierenden DSMS. Da sich die vorliegende Arbeit jedoch mehr auf die Optimierung der Migration konzentriert, wurde lediglich ein einfacher Scheduler im DS-Manager implementiert.

Eingehende Daten werden entsprechend dem FIFO-Prinzip verarbeitet und ein eingehender Wert wird immer vollständig verarbeitet bevor der nächste Eingangswert für die Bearbeitung freigegeben wird. Um festzustellen, wann ein Eingangswert vollständig verarbeitet wurde, besitzen die Systemelemente einen Betriebszustand, der vom DS-Manager überwacht wird. Systemelemente wechseln nach ihrer Verarbeitung in den Betriebszustand „ruhend“. Erst wenn alle Systemelemente diesen Betriebszustand eingenommen haben, wird der nächste Eingangswert freigegeben. Im Unterschied zu aperiodischen Systemen kann in periodischen und schwankungsbeschränkten Systemen meistens auf die Steuerung von Eingangswerten verzichtet werden, da $T_p^* \ll T$ ist und es somit nicht zu konkurrierenden Prozessen kommen kann.

Die Verarbeitungsreihenfolge der Systemelemente ist durch die Komposition der Anfrage, d. h. durch die Verbindungen zwischen den Systemelementen, vorgegeben. Sobald Werte in eine Eingangswarteschlange eingefügt werden, reiht sich der entsprechende

Operator[1] in eine Liste von Systemelementen ein, die auf Verarbeitungszeit warten. Der Betriebszustand wird auf „wartend“ gesetzt. Die Operatoren werden gemäß dieser Reihenfolge ausgeführt. Es sei darauf hingewiesen, dass es durch dieses einfache Scheduling-Konzept bei parallelen Teilanfragen zu Verklemmungen (deadlocks) kommen kann.

Während der Migration wird der DS-Manager vom Update-Manager angewiesen, verschiedene Aufgaben auszuführen. Dazu gehört das Anstoßen des Zustandstransfers bei Zustandssendern. Zustandsempfänger werden durch ihre Zustandstransferwarteschlangen aktiviert. Durch Duplikation (Abschnitt 4.1) können auf einem Verarbeitungsknoten Operatoren erzeugt werden, die einen Datenstrom mit der gleichen DSID produzieren, z. B. Originaloperator und modifizierter Operator während der Migration. Um derartige Operatoren voneinander zu trennen, existiert ein Mechanismus zur Verriegelung [Beh10], der es nur einem Operator erlaubt, seine Ergebnisse in der produktiven Anfrage an Nachfolger zu senden. Das sichere Umschalten der Verriegelung am Ende der Migration oder in manchen Fällen auch schon während der Migration, z. B. bei SemS, wird vom Update-Manager signalisiert und vom DS-Manager durchgeführt.

Algorithmen-Pool Der Algorithmen-Pool beinhaltet eine Sammlung von Algorithmen für die Operatoren, z. B. verschiedene Algorithmen zur Durchschnittsberechnung für Aggregationsoperatoren oder Verbundalgorithmen für Verbundoperatoren. Außerdem sind die Zustandstransferstrategien (OF, LF usw.) im Algorithmen-Pool enthalten.

Um den Algorithmus eines Operators zu aktualisieren, der zur Laufzeit noch nicht im Algorithmen-Pool enthalten ist, kann der Quellcode des Algorithmen-Pools durch den neuen Algorithmus ergänzt werden und anschließend das Bundle zur Laufzeit aktualisiert werden. Bereits instanziierte Algorithmen bleiben während der Aktualisierung des Algorithmen-Pools erhalten. Der Algorithmen-Pool unterscheidet sich in dieser Eigenschaft von allen anderen Bundles und ist deshalb in Abbildung 7.1 grau dargestellt.

Update-Manager Der Update-Manager ist ein optionales Bundle und wird ausschließlich zur Steuerung der Migration und des Zustandstransfers benötigt. Mit Kenntnis der Migrationskonfiguration realisiert er die Umsetzung der globalen Transferstrategie durch Überwachung des Migrationsfortschritts und Auslösen der Aktivitäten über den DS-Manager, z. B. das Erstellen von neuen Systemelementen und Verbindungen oder das Signalisieren zum Umschalten des Verriegelungsmechanismus. Seine Aufgaben im Zusammenwirken mit Zustandssendern und Zustandsempfängern wurden im Detail bereits in Abschnitt 5.4 beschrieben. In verteilten Szenarien sind Update-Manager für den Austausch der Transferdatenströme zwischen Verarbeitungsknoten zuständig.

GUI-Adapter Grafische Benutzerschnittstellen (graphical user interfaces, GUI) können die Interaktion mit dem Datenstromsystem unterstützen und vereinfachen. Typischerweise werden sie für das Konfigurieren von Anfragen, zur Visualisierung von Ergebnissen

[1] „Gemeinsame Warteschlangen“ und „gemeinsame Zustände“ werden nicht unterstützt.

und zur Überwachung des Systems genutzt. Der GUI-Adapter ist ein optionales Bundle, welches im Zusammenwirken mit dem DS-Manager einen Datenaustausch mit Benutzerschnittstellen ermöglicht. Die Benutzerschnittstelle selbst kann im Bundle implementiert sein (Java). Sie kann aber auch eine beliebige externe Anwendung sein und über ein Protokoll Daten mit dem Bundle austauschen.

Es besteht einerseits die Möglichkeit, Konfigurationen und Steuernachrichten an den DS-Manager zu übertragen, beispielsweise optimierte Anfragen oder Anweisungen für die manuelle Umschaltung zwischen Operatorversionen. Andererseits kann sich der GUI-Adapter für beliebige Datenströme innerhalb des Systems registrieren und die empfangenen Daten in geeigneter Form an registrierte Anwenderschnittstellen übertragen. Ebenso können Systeminformationen, z. B. die Betriebszustände der Systemelemente, übermittelt werden. Das Beobachten des Systems stellt in jedem Fall eine Mehrbelastung für CPU und Netzwerk dar, da die Daten für die Übertragung gemäß dem Übertragungsprotokoll paketiert und verteilt werden müssen. In [Wie09] wurde eine Implementierung für die Benutzerschnittstellentechnologie Adobe Flex beschrieben.

Logging Das Logging-Bundle ist ein weiteres optionales Modul und stellt verschiedene Funktionen zur Erfassung von bestimmten Systemdaten zur Verfügung. Es können Datenströme innerhalb des Systems aufgezeichnet und in Dateien abgespeichert werden. Dabei kann die Erfassung alle Datenstromwerte betreffen oder auf bestimmte Datenstromwerte beschränkt werden, beispielsweise die jeweils letzten Ergebnisse von Eingangswerten. Mit dem Logging-Bundle können außerdem Systemaktivitäten protokolliert werden, z. B. das Aktivieren oder Deaktivieren von Operatoren oder das Verhalten bei Zustandstransfers. Das Logging-Bundle wurde hauptsächlich für Performance-Untersuchungen genutzt, kann aber auch generell zur Aufzeichnung der Systemaktivität verwendet werden, beispielsweise zur Erfassung und Diagnose von Fehlern. Durch die Datenerfassung und vor allem durch das Persistieren entsteht eine zusätzliche Last für das System, weshalb eine sinnvolle Konfiguration der Protokollierung erforderlich ist. Details zur Anwendung und Konfiguration für die Performance-Untersuchungen werden im Abschnitt 7.1.3 beschrieben oder gegebenenfalls zum jeweiligen Experiment angegeben.

Zusammenfassung Der Prototyp wurde mit der OSGi-Framework-Implementierung Eclipse Equinox [Eqnx11] umgesetzt. Weitere Details zur Implementierung des Prototyps wurden in [Ros08, Beh10] beschrieben. Neben dem Konzept zur Verriegelung von duplizierten Operatoren werden u. a. auch Ansätze zur automatischen Operator-Versionierung oder zur Vergabe von DSIDs vorgestellt.

7.1.2 Migration im Prototyp

Der Prototyp dient als Experimentierumgebung für die Durchführung von Migrationsexperimenten mit verschiedenen Konfigurationen. Der Schwerpunkt liegt dabei auf der Be-

wertung der Migrationsdurchführung, insbesondere der Nachweis einer korrekten Durchführung und die Bestimmung von Laufzeiteigenschaften verschiedener Konfiguration, z. B. die Migrationsdauer oder die Anzahl der übertragenen Werte. Weniger im Vordergrund steht die Suche der Zielanfrage und der Migrationskonfiguration, da es für die korrekte Durchführung der Migration unerheblich ist, ob die Konfiguration durch das System automatisch bestimmt wird oder vor Experimentbeginn bereits feststeht.

Für die Evaluierung wurden fertig konfigurierte Testfälle erstellt, welche zu Beginn des Experiments in den DS-Manager und den Update-Manager geladen werden. Darin enthalten sind die Konfigurationen der Originalanfrage und der Zielanfrage, unter Umständen auch als Teilanfrage, falls nur einige Operatoren während der Migration modifiziert werden. Außerdem ist die Migrationskonfiguration beschrieben. Das sind in jedem Fall die Zustandspaare und in den meisten Fällen die lokale und globale Transferstrategie. Manche Testfälle errechnen Migrationsparameter zur Laufzeit und die Migrationskonfiguration wird damit entsprechend vervollständigt. Die Zustandspaare wurden manuell festgelegt, da die automatische Suche von Zustandspaaren nicht Thema dieser Arbeit ist. Mögliche Schwierigkeiten einer automatischen Suche, insbesondere bei Anfragen mit beliebigen Operatoren, wurden in Abschnitt 5.1 diskutiert. Die Integration einer vollständig automatischen Suche der Migrationskonfiguration ist eine mögliche Erweiterung für den Prototyp.

Mit Ankunft der Anfragenkonfigurationen im DS-Manager wird zunächst die Originalanfrage angelegt und ausgeführt, gegebenenfalls parallel zu bereits existierenden, laufenden Anfragen. Zu einem zuvor bestimmten Zeitpunkt wird die neue Anfrage oder Teilanfrage instanziiert und alle beteiligten Systemelemente für die Migration initialisiert. Dazu gehören das Konfigurieren des Update-Managers, das Einrichten der Zustandstransferstrategien innerhalb der Zustandssender, das Anlegen der Transferwarteschlangen bei den Zustandsempfängern und schließlich das Verbinden der Zustandssender und -empfänger. Falls es im Experiment vorgesehen ist, dass eine Zustandstransferstrategie oder ein neuer Algorithmus erst während der Laufzeit geladen wird, erfolgt als erster Schritt das Aktualisieren des Algorithmen-Pools.

Unmittelbar nach der Initialisierung beginnt die Migration. Sofern ein Experiment in einem aperiodischen System oder mit der Einstellung „sofortiger Zustandstransfer“ durchgeführt wird, wird umgehend mit dem Zustandstransfer begonnen. In periodischen und schwankungsbeschränkten Systemen ohne sofortigen Zustandstransfer wird gemäß dem numerischen Ansatz auf den oder die nächsten Eingangswerte gewartet und der Zustandstransfer erst nach deren Verarbeitung gestartet.

Vor, während und nach der Migration besteht die Möglichkeit, Laufzeitdaten zu erfassen, um anschließend den Migrationsverlauf zu bewerten. Die Konfiguration für die Datenerfassung ist ebenfalls in den Testfällen gespeichert.

7.1.3 Laufzeiteigenschaften und Protokollierung im Prototyp

Die Verwendung von OSGi bietet u. a. den Vorteil, Anpassungen zur Laufzeit mit Mitteln des OSGi-Frameworks zu realisieren, vor allem das Nachladen von Code. OSGi bringt aber auch den Nachteil mit sich, durch die Ausführung der JVM (Java Virtual Machine) als Systemprozess zumindest in Windows-Betriebssystemen nicht echtzeitfähig zu sein. Ursache ist, dass die Ausführung der JVM durch andere Systemprozesse unterbrochen werden kann, auch dann wenn die JVM mit der höchsten Priorität ausgeführt wird [Boy08a, Hol06]. Die allgemeinen Laufzeiteigenschaften des Prototyps und eine geeignete Konfiguration für die Protokollierung von Ergebnissen und Messdaten zur Bewertung des Systemverhaltens werden in diesem Abschnitt diskutiert.

Performance-Untersuchungen in Java sind generell schwierig, da es viele Einflussfaktoren gibt, die sich auf die Laufzeit des Java-Codes auswirken [Boy08a]. Bereits Tests von einfachsten Java-Klassen (z. B. ein „32-bit linear feedback shift register“ in [Boy08c]) haben gezeigt, dass immer Abweichungen bezüglich der Laufzeit auftreten. Gründe dafür sind nach [Boy08c] u. a. Cache-Verfehlen (cache misses), Multi-Threads und deren Synchronisierung oder die automatische Speicherbereinigung und Finalisierung von Objekten (garbage collection and object finalization). Außerdem können Kontextwechsel (context switches) für Unterbrechungen der Ausführung von wenigen Millisekunden sorgen. Manche dieser Prozesse lassen sich beeinflussen, beispielsweise durch Optionen beim Starten der JVM, andere Prozesse sind jedoch nicht beeinflussbar. Aus diesem Grund können bestenfalls schwankungsbeschränkte Datenströme innerhalb der JVM erzeugt und als solche verarbeitet werden.

Ein weiterer Effekt, den man beim Ausführen einer JVM beobachten kann, ist eine verlangsamte Verarbeitung durch die anfängliche Verwendung eines Interpreters. Erst nachdem eine bestimmte Menge von Objekten erzeugt wurde, wird zur Just-in-time-Kompilierung (JIT-Kompilierung) gewechselt, deren Code wesentlich schneller abläuft. Aus diesem Grund sollten Performance-Messungen immer erst in dieser Phase durchgeführt werden, sofern nicht das Anlaufverhalten des gesamten Systems untersucht wird [Boy08a].

Damit die Laufzeiteigenschaften des Systems während der Migration und somit auch die Eigenschaften der Migrationsstrategien bewertet werden können, müssen die Ankunftszeiten von Datenstromelementen an definierten Punkten protokolliert werden, ohne jedoch das System zu behindern oder zu verlangsamen. Das Protokollieren stellt in jedem Fall eine Mehrbelastung für das System dar. Ziel ist es, diese Last so gering wie möglich zu halten, um sinnvolle Aussagen über die Leistungsfähigkeit der getesteten Migrationsstrategien treffen zu können.

Die Experimente beziehen sich in erster Linie auf die Zeiten im System, z. B. Verarbeitungszeiten, Durchlaufzeiten oder Migrationsdauer. Deshalb besteht die Möglichkeit, aufkommende Datentupel an verschiedenen Stellen im System zu protokollieren, beispielsweise in den Eingangswarteschlangen, im Operator oder beim Verlassen des Operators. Außerdem kann neben dem Erfassen aller Tupel das Protokollieren auf bestimm-

te Datentupel oder Teile von Datentupeln beschränkt werden, z. B. nur das erste und das letzte Ergebnis für ein Eingangstupel bzw. nur ein Zeitstempel statt des gesamten Inhalts eines Datentupels. Reguläre Datentupel und Zustandstransferwerte können getrennt voneinander aufgezeichnet und gespeichert werden. Die verschiedenen Varianten wurden als separate Protokollierungsstrategien im Prototyp implementiert.

Um keine langsamen Schreiboperationen während der Migration zu haben, werden alle Daten im Hauptspeicher[2] erfasst und erst nach Abschluss eines Experiments persistent in eine Datei gespeichert. Als Datenstruktur für die Speicherung von Tupeln und Zeitstempeln ist eine verkettete Liste (insbesondere `ArrayList`) geeignet (siehe auch [Boy08b]), welche im Prototyp Zugriffszeiten von wenigen Nanosekunden aufweist. Um die Aktivitäten der Speicher-Allokation zu minimieren, wird die notwendige Speichergröße vor der Durchführung des Experiments abgeschätzt. Sie ist abhängig von der verwendeten Protokollierungsstrategie und der Experimentlänge. Die verwendeten Listen werden dann in der entsprechenden Größe initialisiert und der notwendige Speicher alloziert.

Zur Zeitmessung stehen in Java mindestens zwei Methoden zur Verfügung. Zum einen kann mit `System.currentTimeMillis()` eine Repräsentation der Systemzeit in Millisekunden ermittelt werden. Diese absolute Zeit hat eine systemabhängige Auflösung, welche für Windows XP 15 ms beträgt [Boy08a]. Zum anderen besteht die Möglichkeit mit der Methode `System.nanotime()` eine relative Zeit in Nanosekunden-Auflösung zu erhalten. Mittels Differenzbildung können dann Laufzeiten einzelner Prozesse bestimmt werden. Da `System.nanotime()` eine höhere Auflösung bietet, ist diese Methode zu bevorzugen, sofern eine Differenzbildung zur Leistungsbeurteilung ausreicht [Hol06]. Aus diesem Grund werden im Prototyp die Ankunftszeiten unter Verwendung dieser Methode protokolliert.

Zur Minimierung der Abweichungen beim Messen der Performance wurden außerdem folgende Richtlinien für die Durchführung der Experimente aus den soeben beschriebenen Eigenschaften von Java abgeleitet. Unter Berücksichtigung der Richtlinien liegt der Anteil der Protokollierung an der Gesamtlast deutlich unter 1 % und ist somit für die Bewertung der Migrationsstrategien vernachlässigbar.

- Während der Ausführung werden für die Messung unnötige Systemprozesse deaktiviert, z. B. Virenscanner oder Firewall.
- Die Ausführung von OSGi auf dem Testsystem (Intel-Core-2-CPU) wird auf eine CPU begrenzt. Die andere CPU übernimmt alle anderen Prozesse.
- Beim Starten von OSGi wird mittels JVM-Option eine hinreichend große Menge Arbeitsspeicher zugewiesen, um Allokationsprozesse zu vermeiden.

[2] Zwar lautete eine der Anforderungen, den notwendigen Speicher möglichst gering zu halten, der Speicherverbrauch wurde jedoch nicht explizit untersucht. Grund dafür ist, dass die entwickelten Migrationsstrategien die Migration nicht in Bezug auf den Speicherbedarf oder für Speicherengpässe optimieren, sondern in Hinblick auf die Migrationsdauer. Deshalb wird für jedes Experiment die Verfügbarkeit von hinreichend viel Arbeitsspeicher garantiert.

- Zum Erreichen des Betriebsmodus mit JIT-Kompilierung wird eine Anlaufzeit definiert, in der keine Experimente durchgeführt werden – üblicherweise einige Minuten.
- Die Protokollierung wird auf die für die Auswertung notwendigen Werte beschränkt. Die Protokollierungsstrategie wird entsprechend ausgewählt.
- Die notwendige Speichergröße zur Erfassung der Protokollierungsdaten werden vorab abgeschätzt und die Datenstrukturen entsprechend initialisiert, um die Speicherallokation während der Laufzeit und übermäßige Speicherbereinigung zu vermeiden.
- Falls möglich, wird vor der Migration mehrmals eine Methode zur Speicherbereinigung und Objekt-Finalisierung ausgeführt, um für unterschiedliche Strategien und Durchläufe ähnliche Bedingungen zu ermöglichen.
- Experimente werden mehrfach wiederholt, um Durchläufe mit eventuellen Ausreißern oder Fehlern zu verwerfen.

Insbesondere für den letzten Punkt ist die Verwendung eines Gütekriteriums notwendig, um gültige von ungültigen Versuchen zu unterscheiden. Wie bereits beschrieben, lassen sich die Datenströme im System als schwankungsbeschränkte Datenströme klassifizieren. Als solche besitzen sie die Parameter τ und τ', welche die potentiellen Abweichungen charakterisieren. Der Parameter τ' beschreibt die maximale Verspätung und ist somit als Vergleichsgröße geeignet. Ist während eines Versuchs ein Eingangswert um mehr als τ' verspätet, dann ist der Versuch ungültig, da die Ursache für die Verzögerung nicht im Prototyp sondern beim Betriebssystem zu suchen ist. Zur Bestimmung eines geeigneten τ' wurden die einzelnen Experimente ohne Migration durchgeführt. Typischerweise ist $\tau' \approx 15\,\text{ms}$.

In Echtzeitsystemen sollte sich τ' deutlich reduzieren lassen, da die vom System verursachten Verzögerungen, falls sie überhaupt existieren, deutlich geringer sein sollten. Eine Lösungsmöglichkeit ist die Erweiterung eines existierenden, echtzeitfähigen DSMS mit einer Technik zum Nachladen von Algorithmen zur Laufzeit sowie das Einrichten der für die Migration und den Zustandstransfer notwendigen Funktionen. Eine andere Realisierung ist ein echtzeitfähiges OSGi. Auf Linux-Systemen besteht die Möglichkeit eine Echtzeit-JVM auszuführen. Eine Spezifikation für ein Echtzeit-OSGi existiert jedoch gegenwärtig nicht und auch wissenschaftliche Prototypen oder proprietäre, kommerzielle OSGi-Systeme mit Echtzeiteigenschaften sind selten [RA10].

7.2 Korrektheit der Transfermethoden

Ein grundlegender Schritt vor der Performance-Bewertung der Experimente ist die Überprüfung auf die Korrektheit ihrer Durchführung, insbesondere in Bezug auf die erzeugten

Ausgabedaten. Dafür werden die protokollierten Ergebnisse genutzt. Da im Verlauf der Migration durch das Umschalten von Original- auf Zielanfrage von diesen beiden Anfragen Ergebnisse produziert werden, ist nachzuweisen, dass keine Fehler im Verlauf der Migration und besonders beim Umschalten aufgetreten sind. Eine korrekte Ausgabe sowie Symptome von Fehlern[3] sind in Abbildung 7.2 aus Sicht eines nachfolgenden Operators dargestellt. Es sind die Tupel-Indizes über der Ausgabezeit aufgetragen[4], wobei zu jedem Zeitpunkt das Tupel mit dem entsprechenden Index erwartet wird, z. B. das Tupel mit dem Index 15 bei $t = 15$. Es sind jeweils die Ausgaben eines Originaloperators und des neuen Operators, der ihn ersetzt, abgebildet. Die korrekte Ausgabe (Abbildung 7.2a) zeigt, dass Werte bis $t = 14$ vom Originaloperator ausgegeben werden und ab $t = 15$ vom neuen Operator. In Abbildung 7.2b ist die Ausgabeverzögerung dargestellt, wie sie beispielsweise bei der Moving-State-Strategie auftritt. Für $14 < t \leq 17$ werden keine Ergebnisse ausgegeben, dafür kommt es im Zeitraum $17 < t < 19$ zu einer Häufung der Ergebnisausgabe, was mit einem Burst-Verhalten gleichgesetzt werden kann. Abbildung 7.2c stellt einen doppelten Wert bei $t = 15$ dar. In Abbildung 7.2d fehlt hingegen ein Wert bei $t = 16$. Eine Form der Ausgabeverzögerung zeigt Abbildung 7.2e. Das Ergebnis von $t = 15$ wird erst nach $t = 16$ ausgegeben, wodurch diese beiden Ergebnisse vertauscht werden. Wenngleich manche Datenstromsysteme in der Lage sind, mit vertauschten Werten umzugehen, wird dies für die nachfolgenden Experimente als Fehler gewertet.

Zur Bewertung der Korrektheit, werden die Ausgangsdatenströme der von der Änderung betroffenen Teilanfrage für einen Eingangsdatenstrom erfasst. Der Protokollierungszeitraum ist die Migration und eine bestimmte Zeit davor und danach. Zum Vergleich werden im Anschluss an ein Experiment jeweils die Ergebnisse der Original- und Zielanfrage für den gleichen Eingangsdatenstrom aber ohne Migration aufgezeichnet. Die Ausgabe des Migrationsexperiments muss vor dem Umschalten der Ausgabe der Originalanfrage und danach der Ausgabe der Zielanfrage entsprechen. Es werden sowohl die Inhalte der Datentupel als auch die relative Ausgabezeit verglichen. Dabei müssen die ermittelten Ergebnisse identisch sein und die relativen Ausgabezeiten dürfen sich nur geringfügig unterscheiden.

An den nachfolgenden Beispielen wird die Überprüfung von Experimenten auf deren Korrektheit verdeutlicht. Das erste Beispiel bezieht sich auf die Migration einer Anfrage mit einem Aggregationsoperator, der einen gleitenden Mittelwert berechnet. Die durchgeführte Änderung ist eine Vergrößerung des Fensters des Mittelwert-Operators, weshalb Original- und Zielanfrage semantisch nicht äquivalent sind. Der Umschaltzeitpunkt ist erreicht, wenn die Anzahl der Werte im neuen Fenster die Fenstergröße des

[3] gemäß den Anforderungen

[4] Einige Diagramme dieses Kapitels stellen diskrete Punkte dar, z. B. einzelne Ereignisse (Datenstromwerte) oder Ergebnisse von Performance-Untersuchungen. Zum besseren Verständnis sind gegebenenfalls die zusammengehörenden diskreten Punkte (Markierungen) durch eine Linie miteinander verbunden, beispielsweise alle Ereignisse eines Datenstroms oder Ergebnisse für eine bestimmte Konfiguration.

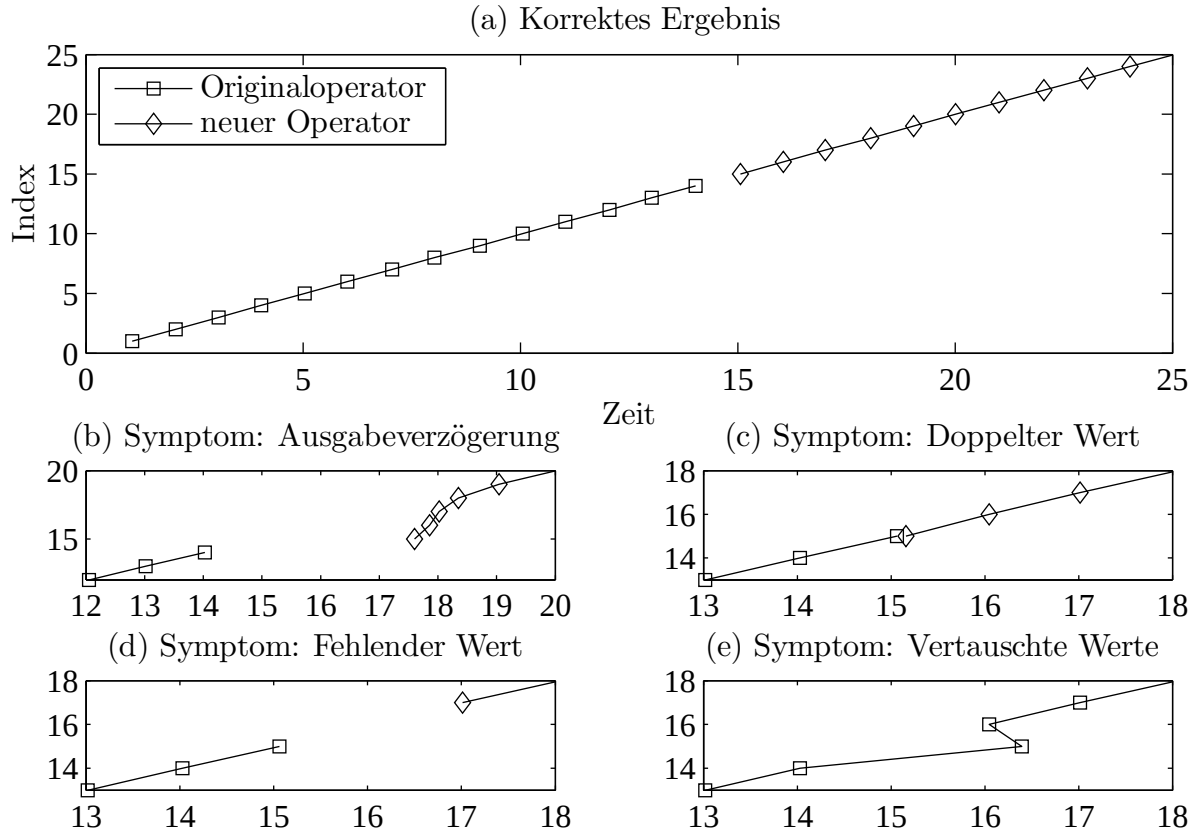


Abbildung 7.2: Symptome für fehlerhafte Ergebnisausgabe während der Migration

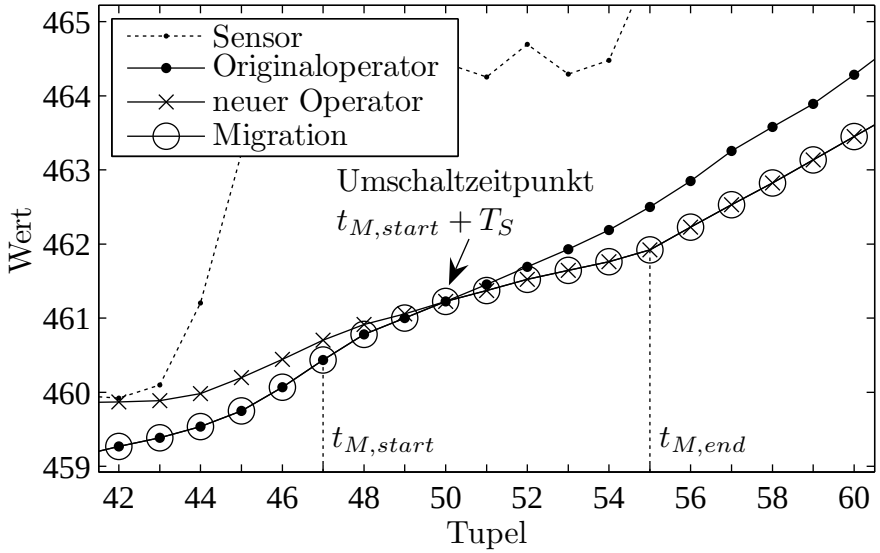


Abbildung 7.3: Bewertung der Korrektheit der Ausgabe vor, während und nach der Migration

Originaloperators übersteigt. In Abbildung 7.3 ist das Ergebnis eines entsprechenden Experiments dargestellt[5]. Neben dem Verlauf des Sensor-Signals, welches mittels Random Walk [Spi01] generiert wurde, sind die Berechnungsergebnisse der Originalanfrage und der neuen Anfrage jeweils ohne Migration eingetragen. Die Resultate der Originalanfrage und der neuen Anfrage liegen aufgrund der unterschiedlichen Fenstergrößen auseinander. Einzige Ausnahme bildet der Umschaltpunkt, bei dem beide Anfragen die gleiche Fenstergröße besitzen. Der Start der neuen Anfrage ist so gewählt, dass deren Anlaufverhalten sich mit der Übergangszeit[6] während der Migration deckt. Zur Bestimmung des Startpunktes werden die errechneten Migrationsparameter (N_n) verwendet. Da für die Durchführung des Experiments mit der neuen Anfrage ohne Migration kein Zustandstransfer verwendet wird, liegt der Beginn auch vor dem Migrationsstart. In aperiodischen Datenströmen ist N_n nach dem Migrationsexperiment zu bestimmen, um den korrekten Startpunkt für den Vergleich mit der neuen Anfrage zu erhalten. Da der gesamte Eingangsstrom vor der Durchführung des Vergleichsexperiments mit der neuen Anfrage bekannt ist, lässt sich der Startpunkt exakt ermitteln. Der Verlauf des Migrationsexperiments zeigt, dass die Ergebnisse bis zum Umschaltzeitpunkt den Ergebnissen der Originalanfrage entsprechen und danach den Ergebnissen der neuen Anfrage. Das Experiment gilt somit als korrekt bezüglich dieser Eigenschaften.

Das zweite Beispiel ist eine Kompositionsänderung eines Verbundbaums, wie es bereits in den vorherigen Kapiteln verwendet wurde (Abbildung 5.7, S. 92). Original- und Zielanfrage sind semantisch äquivalent. Dies bedeutet zwar, dass beide Anfragen für den gleichen Eingangswert die gleichen Ergebnisse produzieren, deren Reihenfolge unterscheidet sich aber aufgrund der unterschiedlichen Kompositionen der beiden Anfragen. Aus diesem Grund wird die Ergebnismenge, deren Größe von der globalen Fenstergröße abhängt, für jeden Eingangswert auf Vollständigkeit überprüft. Vergleicht man die Ergebnisse der Original- und Zielanfrage sowie des Migrationsexperiments, dann müssen in allen drei Fällen die Ergebnismengen zu jedem Eingangswert jeweils identisch sein. Die Ausgabereihenfolge der einzelnen Tupel ist dabei unerheblich. Zur Überprüfung der zeitlichen Korrektheit werden die relativen Ausgabezeiten der jeweils letzten Tupel miteinander verglichen, ob sie innerhalb einer definierten Toleranz liegen. Sind Vollständigkeit und zeitliche Korrektheit gegeben, ist das Migrationsexperiment gültig hinsichtlich der Verbundergebnisse.

Alle durchgeführten und ausgewerteten Experimente, ob simulativ oder im Prototyp, wurden angemessen auf Korrektheit untersucht. Sofern die Ergebnisse nicht korrekt waren, konnten die Fehler stets mit dem Systemverhalten erklärt werden, z. B. Kollisionen infolge von erhöhter Last durch andere Betriebssystemprozesse. Die entsprechenden Experimente wurden für die Auswertung nicht berücksichtigt.

[5] Die Tupelnummern wurden für die Darstellung nachträglich geändert (Reduktion auf Zehner- und Einerstelle).

[6] Zeitintervall zwischen Umschalten zur neuen Anfrage und Erreichen der neuen Fenstergröße, d. h. $[t_{M,start} + T_S, t_{M,end}]$.

7.3 Allgemeine Festlegungen

Für die Durchführung der Migrationsexperimente werden nachfolgende Festlegungen getroffen. Sofern diese nicht zutreffen sollten, beispielsweise bei den existierenden Migrationsstrategien (Abschnitt 7.5), werden Ausnahmen oder Anpassungen an entsprechender Stelle diskutiert. Zwar wurden alle Größen bereits im Verlauf der Arbeit definiert, da sie sich aber zwischen den verschiedenen Ansätzen unterscheiden können, werden die wichtigsten Details insbesondere im Hinblick auf die Evaluierung hier zusammengefasst. Des Weiteren werden die verwendeten Beispielanfragen beschrieben.

Messgrößen Die in dieser Arbeit vorgestellten Migrations- und Zustandstransferstrategien sind Konzepte zur Verbesserung der Migrationsdurchführung. Aus diesem Grund liegt der Schwerpunkt der Evaluierung auf der Migration selbst und nicht auf der Anfrageoptimierung. Für die Ermittlung der Eigenschaften der Migrationsstrategien wurden folgenden Größen durch Experimente im Prototyp oder durch Simulation bestimmt:

- Migrationsdauer T_M – Die gemessene Migrationsdauer ist ein wesentliches Kriterium zur Bestimmung der Qualität einer Migrationsstrategie. Da die Migrationsdauer die Zeitspanne zwischen Migrationsbeginn und Migrationsende ist, hängt ihre Korrektheit von der Definition dieser beiden Werte ab. Nur eine identische Verwendung in allen Experimenten lässt einen Vergleich verschiedener Migrationsverfahren zu. Der Migrationsbeginn markiert den Zeitpunkt des Verarbeitungsbeginns des ersten Wertes nach Auslösen der Migration. Dies kann entweder ein regulärer Wert sein, wie im Fall der periodischen oder schwankungsbeschränkten Datenströme oder ein Zustandstransferwert bei Verwendung des Sofortigen Zustandstransfers oder in aperiodischen Datenströmen. Das Migrationsende ist der Zeitpunkt, zu dem der letzte Wert der Migration verarbeitet wurde. Für den numerischen Ansatz oder bei der Migration eines einzelnen Fensters ist dies stets ein regulärer Wert. Bei der Verwendung des heuristischen Ansatzes kann auch ein Zustandstransferwert die Migration eines Zustandes abschließen.

- Zustandstransferdauer T_{ZT} – Die Zustandstransferdauer lässt sich wesentlich einfacher bestimmen, da sie sich ausschließlich auf Zustandstransferwert bezieht. Sie repräsentiert die Zeitdifferenz zwischen dem ersten und dem letzten Zustandstransferwert.

- Anzahl übertragener Zustandswerte N_o oder $\mathcal{N}_o$ – Bezogen auf einen einzelnen Zustand ist die Anzahl der übertragenen Zustandswerte N_o. Wurden mehrere Zustände transferiert, ist die Summe aller übertragenen Zustandswerte $\mathcal{N}_o$.

- Ergebnisverzögerung T_{OS} – Die Ergebnisverzögerung („Output Silence“) ist die längste Verzögerung zwischen einem Eingangswert und zugehörigem Ergebnis, die

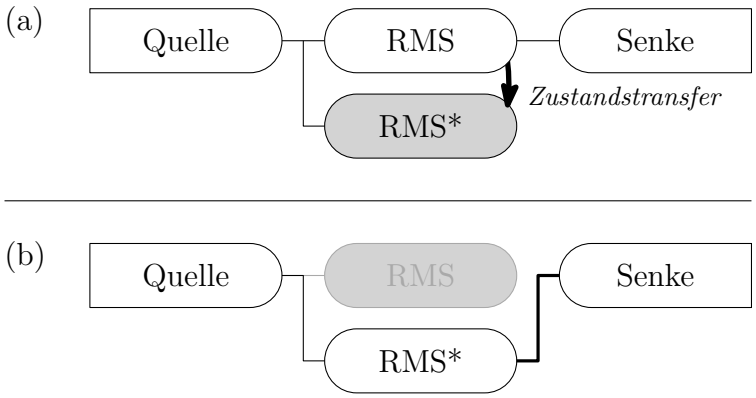


Abbildung 7.4: Beispielanfrage RMS (Root Mean Square, Effektivwert) zur Evaluierung der Zustandstransferstrategien

während der Migration gemessen wird. Wird ein Eingangswert unmittelbar nach dem Eintreffen verarbeitet und seine Ergebnisse ausgegeben, dann ist $T_{OS} = 0$.

Nicht untersucht wurden beispielsweise Speicherverbrauch und Durchsatz von Anfragen, auch wenn dies für existierende Migrationsverfahren getan wurde, z. B. in [YKPS07]. Der Grund dafür ist, dass diese Größen eher der Anfrageoptimierung zuzuordnen und unabhängig von der Migrationsoptimierung sind.

Beispielanfragen Für die Evaluierung wurden drei verschiedene Beispielanfragen verwendet. Zwei stammen aus dem Anwendungsfeld der zustandsorientierten Instandhaltung. Es wurden bewusst einfache, deterministische Anfragen gewählt, um die errechneten Ergebnisse hinsichtlich ihrer Korrektheit miteinander vergleichen zu können und schließlich vergleichbare Werte bezüglich der Migrationseigenschaften zu erhalten.

Die einfachste Anfrage, die für die Untersuchung der entwickelten Strategien verwendet wurde, ist die *Beispielanfrage RMS*, dargestellt in Abbildung 7.4. Diese berechnet mit Hilfe eines Aggregationsoperators einen speziellen gleitenden Mittelwert, nämlich den Effektivwert (RMS, von engl. „root mean square“) eines schwingenden Eingangssignals, z. B. einer Spannung oder einer Vibration. Praxisbezogene Beispiele wurden u. a. in [KS03, WRBK01] gezeigt. Die Beispielanfrage RMS steht stellvertretend für andere Aggregationsfunktionen, mit denen die Migrationsstrategien getestet wurden, z. B. andere gleitende Mittelwerte, Minimum oder Maximum.

Für die Berechnung eines Ergebnisses werden Datentupel des Eingangsdatenstroms ($\mathcal{D}_I$) über ein zeitbasiertes oder anzahlbasiertes, gleitendes Fenster im RMS-Operator erfasst. Die Formel zur Berechnung des Effektivwertes lautet

$$rms = \sqrt{\frac{1}{n}\sum_{i=1}^{n} s_i^2} \tag{7.1}$$

wobei s_i dem Signalwert (a) des i-ten Datenstromelements innerhalb des Fensters ($w = n$) entspricht, d. h. $s_i = \mathcal{D}_{I,i}.a$ $(1 \leq i \leq n)$, $\forall i, n \in \mathbb{N}^+$. Ein Ergebnistupel $\mathcal{D}_O$ wird dann jeweils aus dem errechneten Ergebnis und dem Zeitstempel des neuesten Eingangsdatentupels gebildet, d. h. $\mathcal{D}_O = \langle rms, \mathcal{D}_{I,n}.ts \rangle$ und über den Ausgangsdatenstrom ausgegeben. Es besteht die Möglichkeit, dem Datentupel weitere Daten hinzuzufügen, z. B. für die Versuchsauswertung.

Der Migrationsablauf ist in Abbildung 7.4 dargestellt. Die existierende Anfrage wird um einen neuen Operator (RMS*) ergänzt, der sowohl Eingangswerte als auch die übertragenen Zustandswerte aufnimmt (Abbildung 7.4a). Da während der Migration keine Verarbeitung im neuen Operator notwendig ist, können für diesen einfachen Fall die Migrationsparameter unter Verwendung von $T_p = 0$ (für Transferwerte) bestimmt werden. Den Endzustand der Migration zeigt Abbildung 7.4b.

Eine Erweiterung dieser einfachen Anfrage ist die *Beispielanfrage CF*, welche den Scheitelfaktor (Crest-Faktor (CF) [Fri02]) ermittelt. Dieser wird gemäß der nachfolgenden Gleichung berechnet.

$$cf = \frac{\max_i(|s_i|)}{rms} \tag{7.2}$$

Die Beispielanfrage ist in Abbildung 7.5a dargestellt. Der RMS-Operator des vorherigen Beispiels wurde ergänzt durch einen weiteren Aggregationsoperator, der den Maximalwert bestimmt, und einen zustandsfreien CF-Operator, der lediglich die letzten Ergebnisse seiner Vorgängeroperatoren verrechnet. Der Nachteil dieser Komposition ist, dass der RMS-Operator und der Maximumoperator einen identischen inneren Zustand besitzen, was nicht besonders speichereffizient ist. Deshalb werden alle Operatoren der Anfrage mittels Migration durch einen einzelnen, zustandsbehafteten CF-Operator (CF*) ersetzt, wodurch sich der Speicherbedarf halbiert[7]. Der Zustandstransfer findet zwischen dem Maximumoperator und dem neuen CF-Operator statt. Alternativ kann der RMS-Operator als Zustandssender genutzt werden. Abbildung 7.5b repräsentiert die Anfrage nach Abschluss der Migration.

Die dritte Anfrage, *Beispielanfrage Verbundbaum*, repräsentiert eine Gruppe von semantisch äquivalenten Verbundbäumen und wird für die Evaluierung der Migration mit mehreren Zuständen genutzt. Beispiele sind in den Abbildungen 3.1 (S. 31) und 5.7 (S. 92) veranschaulicht. Dies entspricht Migrationsszenarien, wie sie von existierenden Migrationsstrategien genutzt wurden [ZRH04, YKPS07].

7.4 Vergleich der Transfermethoden

Dieser Abschnitt beschreibt die Untersuchung der Migrationseigenschaften der entwickelten Zustandstransfermethoden. Die Analyse dient hauptsächlich dazu, die korrekte Funktionsweise der Zustandstransferstrategien zu überprüfen. Für die Durchführung der

[7] nach Abschluss der Migration

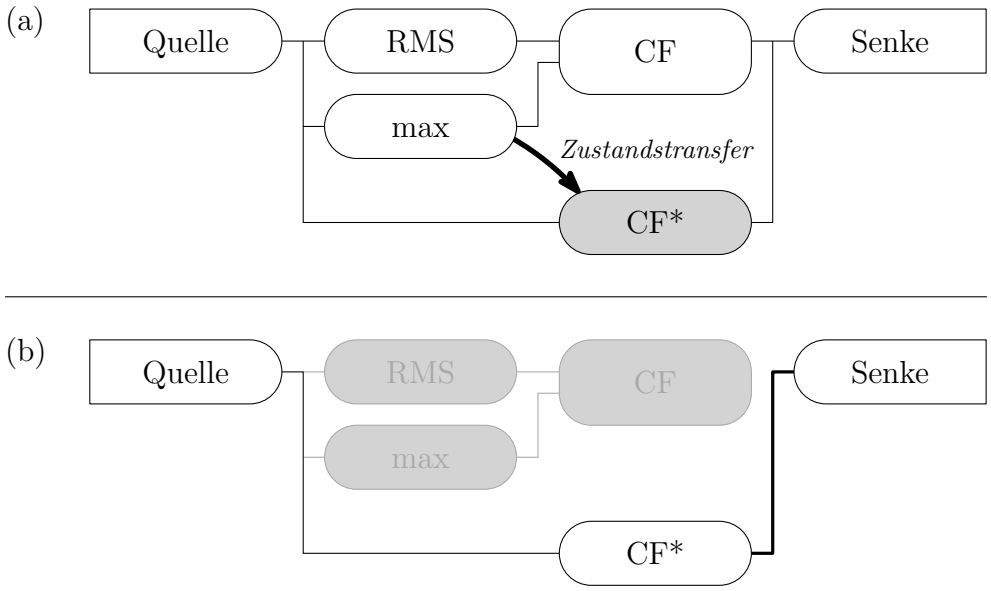


Abbildung 7.5: Beispielanfrage CF (Crest-Faktor, Scheitelfaktor) zur Evaluierung der Zustandstransferstrategien

Experimente wurden die beiden Beispielanfragen RMS und CF genutzt. Jede Methode wurde einzeln für verschiedene Konfigurationen untersucht. Die dargestellten Fälle entsprechen einer Konfiguration für eine Fenstervergrößerung von $w_o = 250$ nach $w_n = 300$, bei einem Zustandstransfer mit $N_t = 8$. Um den Migrationsfortschritt und insbesondere die Transferreihenfolge abzubilden, wurden die Datentupel mit Indizes versehen. Es werden einerseits die Indizes der Tupel über ihrer jeweiligen Verarbeitungszeit dargestellt. Dafür wird das in Abbildung 7.6 skizzierte Schema[8] verwendet. Wird ein Eingangstupel empfangen, so wird es verarbeitet und ein Ergebnis entweder durch den Originaloperator oder durch den neuen Operator ausgegeben. Die Ausgabezeiten der Ergebnisse werden jeweils durch einen Punkt dargestellt, wobei die Operatoren durch unterschiedliche Grautöne repräsentiert werden. Zum neuen Operator wird umgeschaltet wenn die Anzahl der Werte im neuen Fenster die originale Fenstergröße übersteigt. Für die einzelnen Zustandstransferstrategien wird die Abbildung ergänzt durch den Verlauf des Zustandstransfers, symbolisiert durch schwarze Punkte (nicht in Abbildung 7.6).

Jedes Diagramm enthält zusätzliche Linien, welche folgende Bedeutung haben. Zwei vertikale, gestrichelte Linien repräsentieren den Migrationsbeginn (links, ohne Bezeichner) und das Migrationsende (rechts, t_6). Die stufenförmige Linie parallel zu den Ergebnissen markiert das jeweils älteste Tupel im Zustand des Originalfensters zu diesem Zeitpunkt. Eine Zustandstransferstrategie kann also niemals einen Wert übertragen, der unterhalb dieser Linie liegt. Die horizontale Strecke zwischen Migrationsbeginn und

[8] Für eine bessere Übersichtlichkeit sind hier $w_o = 25$ und $w_n = 30$ bei einer Migration ohne Zustandstransfer.

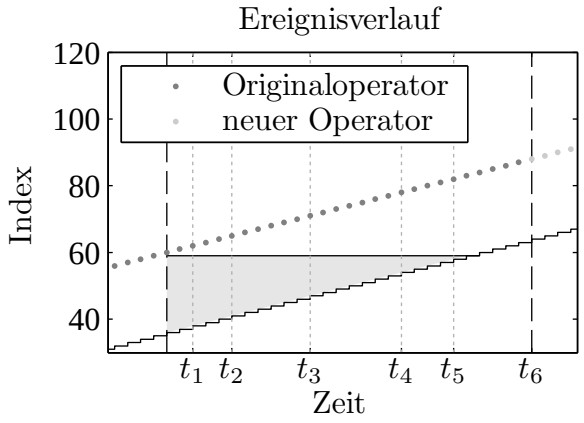


Abbildung 7.6: Schema zur Evaluierung der Zustandstransferstrategien

Erreichen der unteren Grenzlinie markiert das neueste Tupel im Fenster des Originaloperators, welches im neuen Operator nicht bekannt ist, d. h. das letzte reguläre Tupel vor Migrationsbeginn. Eine Migrationsstrategie muss somit keinen Wert übertragen, der oberhalb dieser Linie liegt, da dieser als Eingangswert vom neuen Operator empfangen wird. Das grau schattierte Dreieck stellt folglich den potentiellen Zeitraum für den Zustandstransfer und die verfügbare Anzahl von Zustandswerten dar. Weitere vertikale Linien (t_1 bis t_5) markieren die Zeitpunkte, zu denen ein Abbild des neuen Fensters festgehalten wurde. Die Zeitpunkte sind für jedes Experiment identisch (außer bei RS und SemS). Die Fensterbeispiele sind für die einzelnen Zustandstransferstrategien rechts neben dem Migrationsverlauf abgebildet (z. B. Abbildung 7.7). Es werden für jeden Zeitpunkt alle im neuen Fenster enthaltenen Indizes dargestellt, schwarz wenn es sich um Zustandstransferwerte ($ts < t_{M,start}$) handelt oder grau wenn es sich um reguläre Eingangswerte ($ts \geq t_{M,start}$) handelt. Die Länge vom linken Rand bis $t_{M,start}$ entspricht dabei der Fenstergröße des Originaloperators. Können Werte nicht übertragen werden, bleiben die entsprechenden Stellen weiß. Der rechte Rand repräsentiert das Migrationsende.

OF – Oldest First Der Ereignisverlauf für die Zustandstransferstrategie OF ist in Abbildung 7.7 dargestellt. Wie beschrieben sind die Indizes[9] über der Zeit[10] abgebildet. Der Zustandstransfer beginnt beim ältesten Wert und endet mit dem neuesten Wert der Zustandstransfermenge. Der Ausschnitt zeigt eine Sequenz von acht Transferwerten. Während der Lücken vor und nach der Sequenz pausiert der Transfer zum Empfang und zur Verarbeitung eines Eingangsdatentupels. Anhand der Ausgabeereignisse ist zu sehen,

[9] Für die Darstellung wurden für alle Experimente die Indizes unter Erhaltung der ursprünglichen Reihenfolge nachträglich normalisiert, d. h. der erste Wert der Migration besitzt immer den Index 600. Dadurch können alle Zustandstransferstrategien einfach miteinander verglichen werden.

[10] Damit sich die Zustandstransferstrategien vergleichen lassen, ist der Ereignisverlauf jeweils für den gleichen Zeitabschnitt dargestellt (außer bei RS).

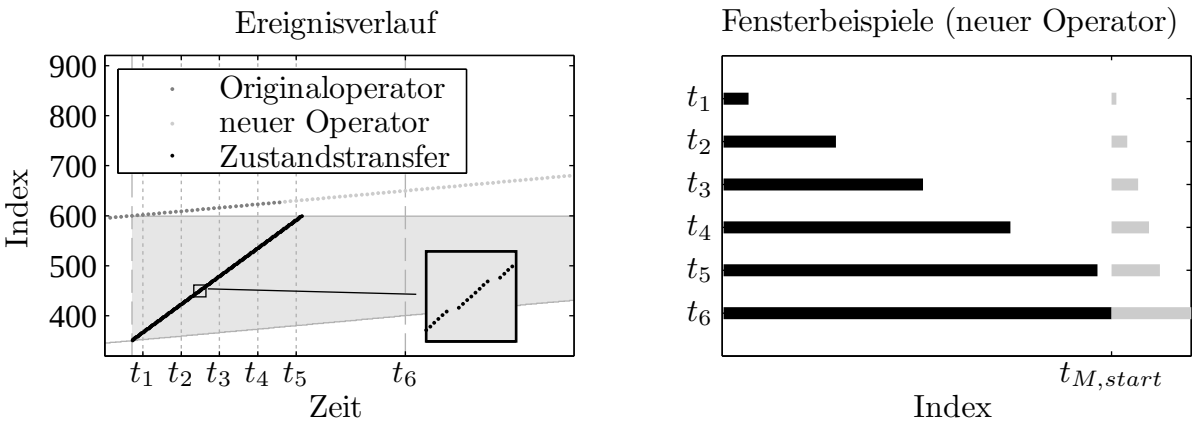


Abbildung 7.7: Verlauf des Zustandstransfers mit OF und Zusammensetzung des neuen Fensters zu verschiedenen Zeitpunkten ($t_6 = t_{M,end}$)

dass das Umschalten zur Ergebnisausgabe des neuen Operators bereits während des Zustandstransfers stattfindet (zwischen t_4 und t_5). Nach Ablauf des Zustandstransfers (nach t_5) werden weiter neue Werte in das Fenster eingefügt bis w_n erreicht ist wodurch die Migration endet (t_6).

Die Fensterbeispiele zeigen, wie sich das Fenster des neuen Operators langsam mit neuen Tupeln und Zustandstransferwerten füllt. Dabei existiert eine Lücke zwischen diesen beiden Mengen bis zum Ende des Zustandstransfers (zwischen t_5 und t_6).

LF – Latest First Mit der Strategie LF werden die Zustandswerte im Vergleich zu OF in der umgekehrten Reihenfolge übertragen (Abbildung 7.8). Es wird mit dem neuesten Wert der Zustandstransfermenge begonnen. Der Transfer endet, wenn kein älterer Wert im originalen Fenster verfügbar ist. Da mit dieser Strategie Werte im alten Fenster verworfen werden, bevor sie übertragen werden können, endet der Zustandstransfer für den dargestellten Fall eher (noch vor t_5) verglichen mit OF. Dies bedeutet auch, dass mehr neue Werte zum Auffüllen des neuen Fensters benötigt werden, weshalb das Migrationsende (t_6) deutlich später erreicht wird.

Dies ist auch bei der Betrachtung der Fensterbeispiele zu beobachten. Durch die fehlenden Werte aus dem alten Fenster bleibt ein Teil der potentiellen Transfermenge frei (linker Rand). Die Menge der neuen Werte ist genau um diesen Anteil größer im Vergleich zu OF. Ein weiterer Unterschied zu OF ist die zusammenhängende Menge von Zustandstransferwerten und neuen Werten. Dieser Aspekt wurde bereits bei der Beschreibung der Zustandstransferstrategien diskutiert (Abschnitt 5.3.1).

Um die Problematik der verworfenen Zustandswerte zu beheben, wurde eine additive Komponente vorgestellt welche sich mit LF zur Zustandstransferstrategie LF+ kombinieren lässt. Deren Migrationsverlauf befindet sich in Abbildung 7.9. Zu beachten sind die einzelnen Transferwerte am unteren Rand. Es werden also jeweils der älteste Wert

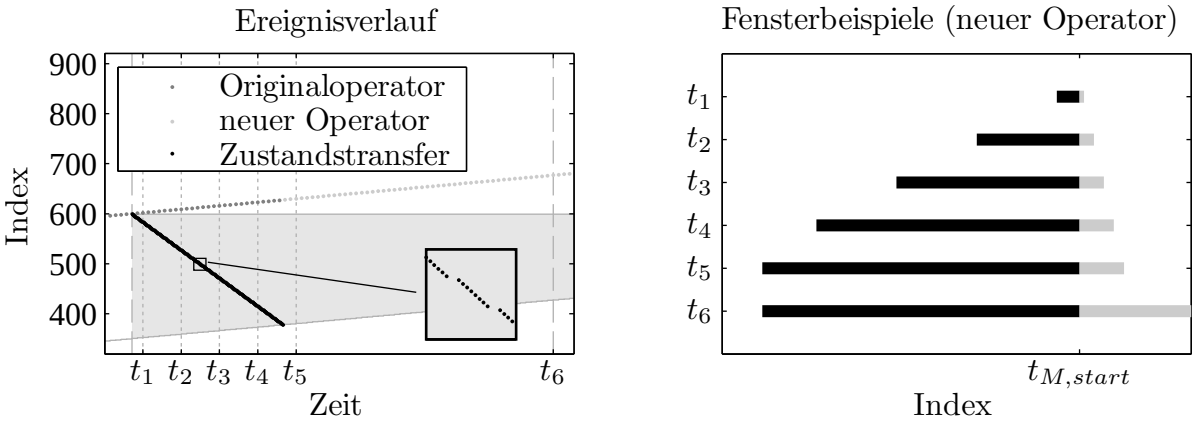


Abbildung 7.8: Verlauf des Zustandstransfers mit LF und Zusammensetzung des neuen Fensters zu verschiedenen Zeitpunkten ($t_6 = t_{M,end}$)

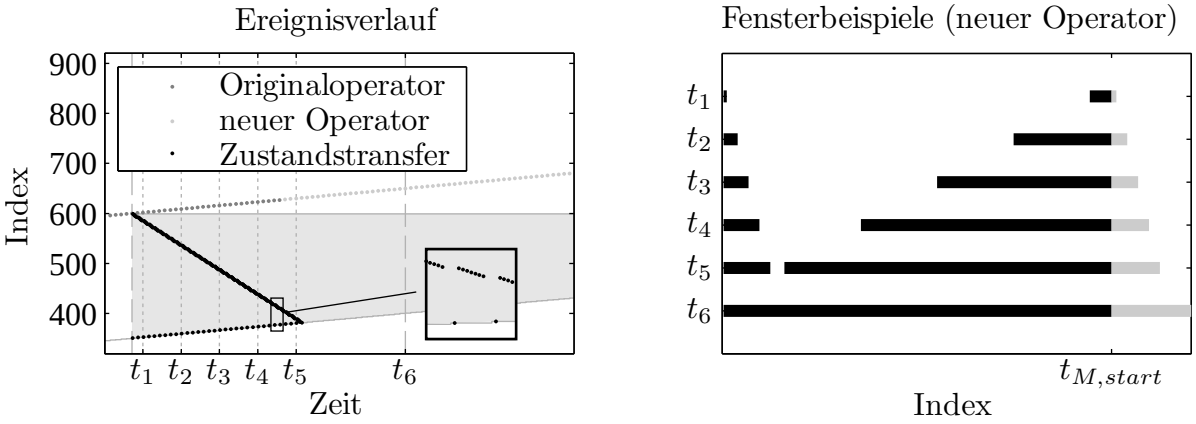


Abbildung 7.9: Verlauf des Zustandstransfers mit LF+ und Zusammensetzung des neuen Fensters zu verschiedenen Zeitpunkten ($t_6 = t_{M,end}$)

der Transfermenge und sieben Werte gemäß LF übertragen. Der Ausschnitt verdeutlicht dies für ca. zwei Transferzyklen. Die Zeitpunkte für das Ende des Zustandstransfers und der Migration sind jeweils identisch zu denen bei OF.

Die Fensterbeispiele zeigen, dass keine alten Werte verloren gehen. Die Lücke zwischen den beiden Mengen verschiebt sich mit fortlaufender Migration in Richtung der ältesten Werte, was für Näherungen nützlich sein kann, die stark auf aktuelle Werte fokussiert sind und neuen Werten ein höheres Gewicht geben. Abgesehen von der Übertragungsreihenfolge der Zustandswerte und der daraus resultierenden Konsequenzen für die Interpretation der Ergebnisse weist LF+ die gleichen Migrationseigenschaften wie OF auf.

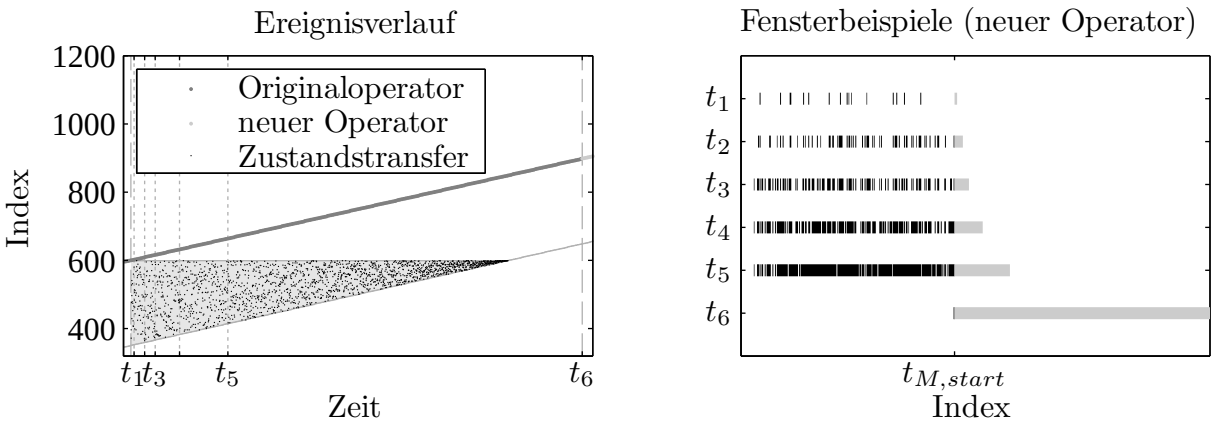


Abbildung 7.10: Verlauf des Zustandstransfers mit RS und Zusammensetzung des neuen Fensters zu verschiedenen Zeitpunkten ($t_6 = t_{M,end}$)

RS – Random Selection Mit der Strategie RS werden Werte aus der Zustandstransfermenge in zufälliger Reihenfolge übertragen. Ob ein Wert bereits ausgewählt wurde oder nicht, wird dabei nicht berücksichtigt. Der Ereignisverlauf (Abbildung 7.10) stellt dieses Verhalten dar. Über den gesamten möglichen Zeitraum werden Werte transferiert, für die verwendete Konfiguration ($w_o = 250$, $w_n = 300$, $N_t = 8$) insgesamt 1992 Datentupel. Da man den Transfer nur für den neuesten Wert der Transfermenge garantieren kann[11], werden $w_n - 1$ neue Werte benötigt, um die Migration zu beenden, setzt man ein lückenlos und vollständig gefülltes, neues Fenster als Abschlussbedingung der Migration voraus. Bereits der zweitälteste Wert kann im ungünstigsten Fall niemals ausgewählt werden, wodurch das Fenster eine Lücke aufweist. Ist kein vollständig gefülltes Fenster zu Vollendung der Migration erforderlich muss eine andere Bedingung definiert werden, beispielsweise eine feste Zustandstransferdauer.

Die Fensterbeispiele verdeutlichen, wie schon zu Anfang der Migration Werte aus der gesamten Breite der Transfermenge im neuen Fenster vertreten sind. Dadurch lässt sich unter Umständen schon frühzeitig eine Näherung über das gesamte Fenster ermitteln, z. B. bei der Berechnung eines gleitenden Mittelwertes eines tendenziell steigenden Sensorsignals. Mit Fortschreiten der Migration werden immer mehr Lücken ausgefüllt. Trotzdem bleiben bis zum Ende des Zustandstransfers Lücken bestehen. Die entsprechenden Werte wurden verworfen, bevor sie wenigstens einmal übertragen werden konnten. Dies ist insbesondere bei den älteren Werten sehr wahrscheinlich. Ein weiterer Effekt der im Verlauf der Migration eintritt, ist das Verwerfen von transferierten Werten im neuen Zustand. Bei Migrationsende ist nur noch der neueste Transferwert im Fenster enthalten. Verglichen mit den anderen Transferstrategien verläuft die Migration äußerst ineffizient und dauert dadurch deutlich länger. Deshalb wird RS nachfolgend nicht weiter

[11] Da im letzten Transferzyklus nur dieser Wert in der Transfermenge zur Verfügung steht wird er in diesem Zyklus genau N_t Mal übertragen.

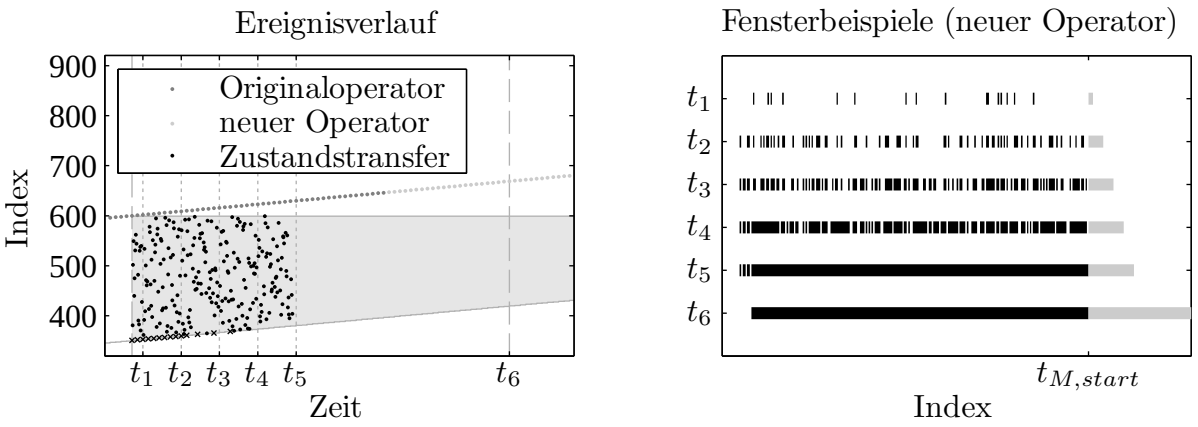


Abbildung 7.11: Verlauf des Zustandstransfers mit RSD und Zusammensetzung des neuen Fensters zu verschiedenen Zeitpunkten ($t_6 = t_{M,end}$)

betrachtet.

RSD – Random Selection without Doubles Zur Kompensation der Nachteile von RS wurde die Zustandstransferstrategie RSD entwickelt. Diese registriert alle übertragenen Werte, sodass kein Zustandswert mehrmals für die Übertragung ausgewählt wird. In Abbildung 7.11 ist der Migrationsverlauf dargestellt. Es wird deutlich, dass der Zustandstransfer deutlich kürzer als bei RS ist. Ein wesentlicher Nachteil von RS bleibt jedoch erhalten. Es gibt Zustandswerte, die verworfen werden, ohne übertragen worden zu sein. Im Ereignisverlauf sind diese Werte jeweils durch ein Kreuz am unteren Rand der Zustandstransfermenge markiert. Im dargestellten Beispiel sind u. a. die ältesten 11 Werte davon betroffen. Insgesamt fehlen 14 Werte.

Durch die fehlenden Werte entstehen Lücken im Bereich der ältesten Werte. Der größte Teil des Fensters kann jedoch lückenlos aufgefüllt werden, weil durch das Erfassen der bereits übertragenen Werte die Transfermenge fortlaufend verkleinert wird, bis schließlich alle Werte, abgesehen von den verworfenen Werten, übertragen wurden. Die Fensterbeispiele verdeutlichen diese Arbeitsweise. Es ist zu beobachten, dass die Migrationsdauer für RSD zwischen denen von LF und OF liegt, d. h. $T_{M,LF} \leq T_{M,RSD} \leq T_{M,OF}$. Eine genaue Migrationsdauer lässt sich unter der Bedingung, ein lückenlos und vollständig gefülltes, neues Fenster zu haben, nicht festlegen, da sie von der zufälligen Auswahl der Reihenfolge der Transferwerte abhängig ist.

Mit der Kombination von RSD mit der Erweiterung zum Erhalt der ältesten Werte ergibt sich die Strategie RSD+. Ihr Verhalten zeigt Abbildung 7.12. Zu Beginn jedes Transferzyklus wird der jeweils älteste Wert der Transfermenge, der bisher nicht übertragen wurde, zum neuen Zustand gesendet. Die verbleibende Transferkapazität wird für zufällig ausgewählte Werte genutzt. Damit kann jeder Wert der Transfermenge übertragen werden. Die Migrationsdauer und die Anzahl der übertragenen Werte entsprechen

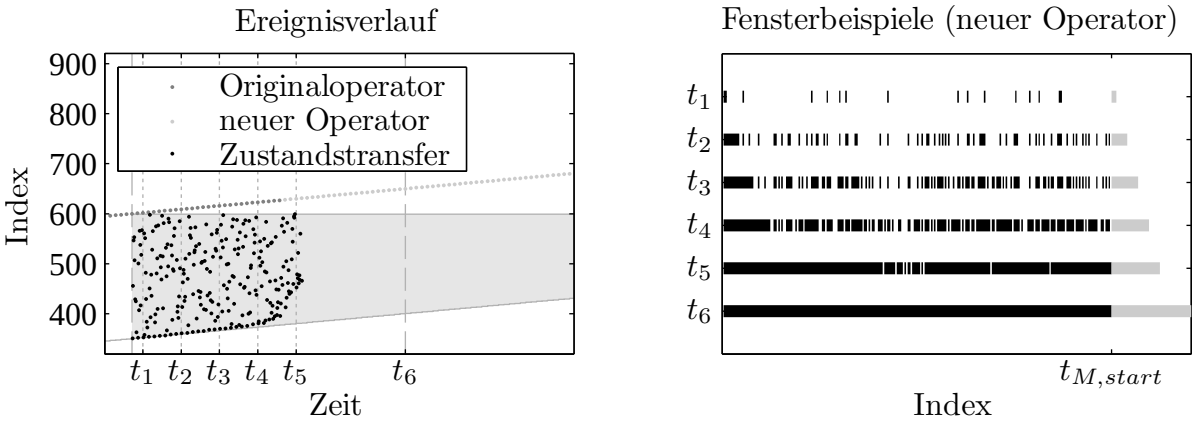


Abbildung 7.12: Verlauf des Zustandstransfers mit RSD+ und Zusammensetzung des neuen Fensters zu verschiedenen Zeitpunkten ($t_6 = t_{M,end}$)

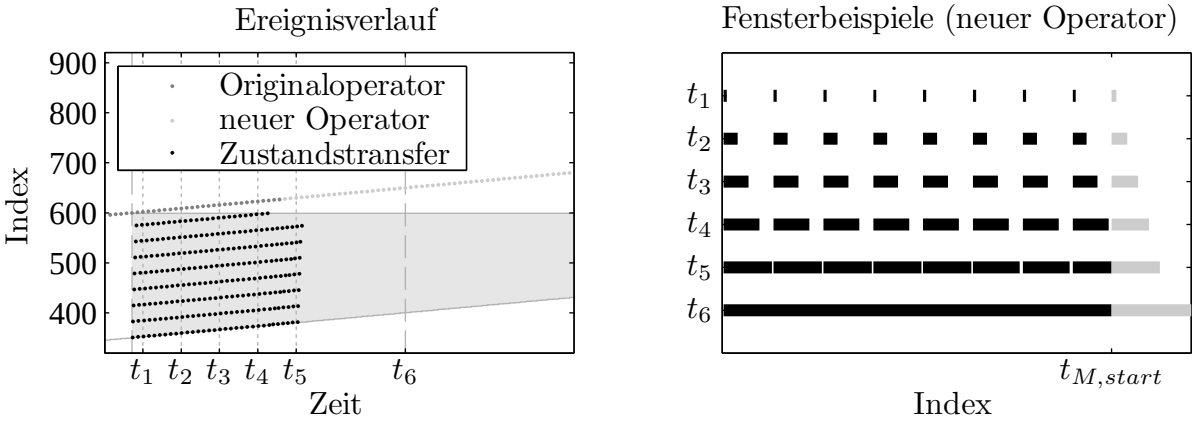


Abbildung 7.13: Verlauf des Zustandstransfers mit DS und Zusammensetzung des neuen Fensters zu verschiedenen Zeitpunkten ($t_6 = t_{M,end}$)

den Eigenschaften von OF. Lediglich die Übertragungsreihenfolge unterscheidet sich. Anhand der Fensterbeispiele ist zu erkennen, wie die Lücken allmählich verschwinden und wie die Transfermenge vollständig übertragen wird.

DS – Distributed Selection Mit Hilfe der Zustandstransferstrategie DS lassen sich die Werte der Transfermenge in einer systematischen Reihenfolge übertragen. Abbildung 7.13 repräsentiert ein entsprechendes Beispiel. Der Abstand der Werte wurde auf $\lceil N_o/N_t \rceil$ festgelegt. Dies ist ausreichend, da im konkreten Beispiel $N_i = 1$ ist. Für $N_i > 1$ muss DS+ für eine verlustfreie Migration verwendet werden. Die Fensterbeispiele verdeutlichen, wie durch die Übertragung Abschnitte entstehen, die im Verlauf der Migration wachsen, bis alle Lücken geschlossen sind. Da die Transfermenge vollständig übertragen wird, entspricht die Migrationsdauer der von OF.

SemS – Semantic Selection Wie in Abschnitt 5.3 bereits beschrieben wurde, ist die Strategie SemS nicht für jeden Anwendungsfall geeignet. Migrationsszenarien mit einzelnen Aggregationsoperatoren sind aber realisierbar, z. B. die Berechnung von gleitenden Summen oder Mittelwerten oder die Ermittlung von Minimum- und Maximumwerten. Für die Evaluierung der Strategie wurden die Beispielanfragen RMS und CF benutzt. Außerdem wurde eine Anfrage mit einen einzelnen Maximumoperator verwendet. Diese wurde für die Dokumentation der Evaluierungsergebnisse von SemS gewählt, da die von der Anfrage ermittelten Ergebnisse relativ einfach nachzuvollziehen sind.

Als Eingangssignal dient ein durch Random Walk erzeugtes künstliches Sensorsignal. Bei Auslösen der Migration werden durch SemS im Zustand des Zustandssenders die Zustandstransferwerte identifiziert. Zwei Beispiele für die Zustandsauswahl bei der Migration mit SemS für die Fenstergrößen $w_o = w_n = 15$ zeigt Abbildung 7.14. Es ist jeweils das generierte Sensorsignal dargestellt. Die durch SemS ausgewählten Werte sind durch einen Kreis markiert und nur diese Werte wurden beginnend mit dem ältesten Wert übertragen. Mit ihnen wird der Zustand im neuen Operator rekonstruiert. Nicht übertragene Werte werden entweder ignoriert (zeitbasierte Fenster) oder durch geeignete Scheinwerte repräsentiert (anzahlbasierte Fenster).

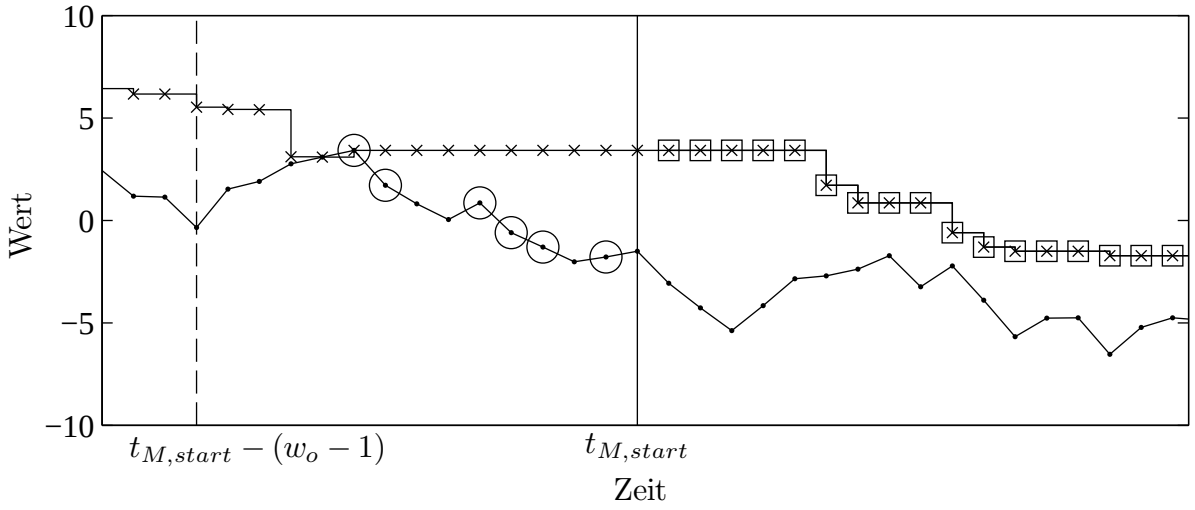
Die ausgegebenen Ergebnisse des originalen und des neuen Operators sind ebenfalls dargestellt. Zum Nachweis der Korrektheit der Ergebnisse des neuen Operators wurde die Ausgabe des Originaloperators auch nach dem Umschalten zum neuen Operator fortgeführt. Es ist zu erkennen, dass der neuen Operator bereits nach einem Transferzyklus in der Lage ist, das korrekte Ergebnis auszugeben.

Eine besondere Situation verdeutlicht Abbildung 7.14b. Hier umfasst der Zustandstransfer nur einen Wert. Dies ist immer dann der Fall, wenn der neuste Wert im Zustand gleichzeitig der größte Wert ist und somit alle anderen Werte dominiert. Diese Situation wird im weiteren Verlauf mit „only 1“ bezeichnet.

Wie für die anderen Zustandstransferstrategien ist der Ereignisverlauf über den gleichen Zeitabschnitt dargestellt (Abbildung 7.15). Es wird jedoch keine Fenstervergrößerung durchgeführt, sondern zwischen gleichgroßen Fenstern mit $w_o = w_n = 250$ migriert. Da die Migration verglichen zu den anderen Strategien deutlich eher endet, entspricht t_6 nicht dem Migrationsende, sondern t_3. Auch die Zeitpunkte der ersten Fensterbeispiele (t_1 bis t_4) wurden auf die ersten regulären Werte nach Beginn der Migration verlegt. Lediglich t_5 und t_6 entsprechen den vorherigen Zeitpunkten.

Sowohl die Transferwerte beim Ereignisverlauf als auch in den Fensterbeispielen verdeutlichen, wie wenige Werte für die die Rekonstruktion notwendig sind und wie viele Werte für den Transfer vernachlässigt werden können. Wiederholt man die Migration für eine Konfiguration hinreichend oft, lässt sich bestimmen, wie viele Werte durchschnittlich oder maximal übertragen werden und wie oft die Situation eintritt, dass nur ein Wert ausgewählt wird. Für beide zuvor genutzten Konfigurationen ist eine entsprechende Analyse einer Simulation der Werteauswahl über 100000 Wiederholungen in Abbildung 7.16 dargestellt. Es ist jeweils abgebildet, wie oft eine bestimmte Anzahl von Werten ausgewählt wurde, normalisiert auf die gesamte Anzahl von Experimenten. Die

(a) SemS(max) für $w_o = 15$



(b) SemS(max) für $w_o = 15$, Beispiel 'only 1'

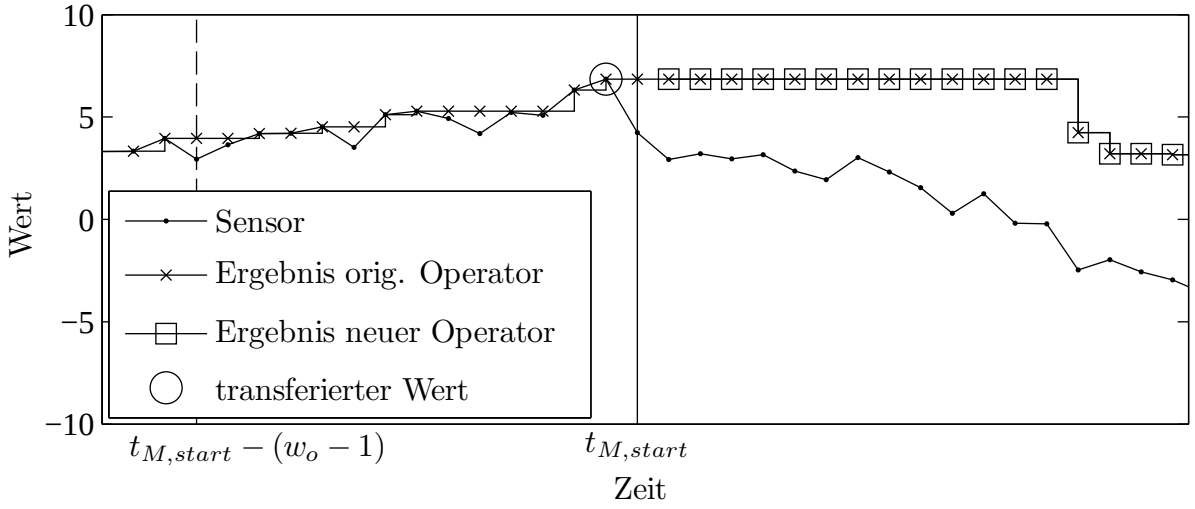


Abbildung 7.14: Beispiele für die Zustandsauswahl für SemS bei Maximum-Operatoren

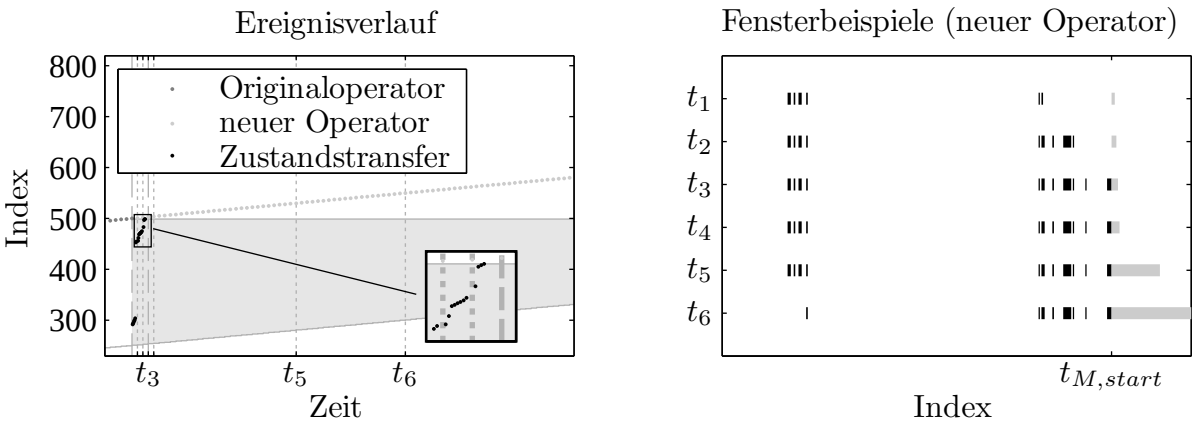


Abbildung 7.15: Verlauf des Zustandstransfers mit SemS und Zusammensetzung des neuen Fensters zu verschiedenen Zeitpunkten

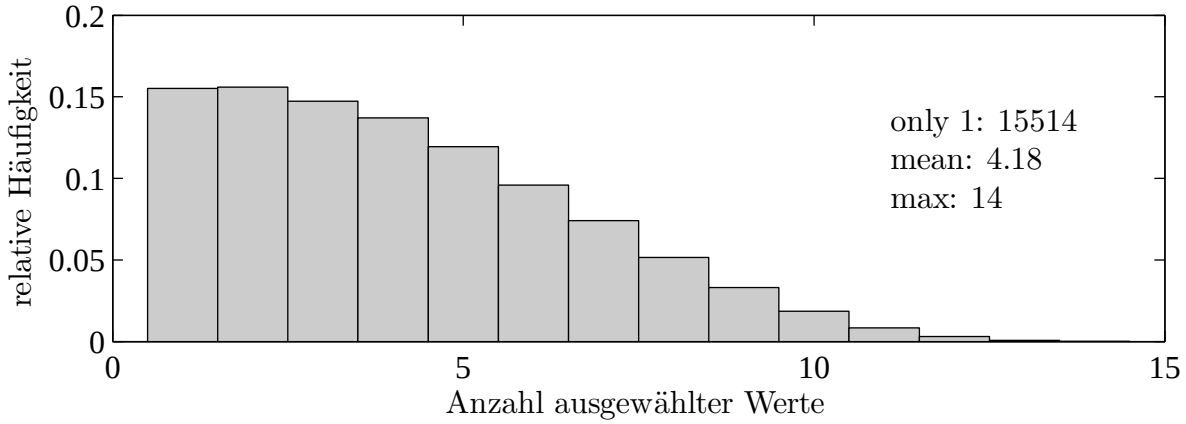
Situation „only 1“ entspricht dabei immer dem ersten Balken eines Histogramms. Die konkrete Anzahl ist jeweils rechts im Diagramm angegeben. Sie wird ergänzt durch die durchschnittliche und maximale Anzahl ausgewählter Werte („mean“ und „max“).

Um zu verstehen, wie sich die Eigenschaften eines Sensorsignals auf die Anzahl von ausgewählten Werten auswirkt, wurde für weitere Experimente der Random Walk zusätzlich mit einer systematischen Abweichung (*bias*) versehen, die das künstliche Sensorsignal tendenziell steigen ($bias > 0$) oder fallen ($bias < 0$) lässt. Abbildung 7.17 zeigt die Ergebnisse für $bias = 0{,}2$ und $bias = -0{,}2$. Alle anderen Parameter wurden nicht verändert.

Für das tendenziell steigende Signal ($bias = 0{,}2$) werden durchschnittlich weniger Werte als zuvor ($bias = 0$, Abbildung 7.16b) ausgewählt. Im ungünstigsten Fall der 100000 Wiederholungen wurden 46 Zustandswerte übertragen, was weniger als einem Fünftel der Fenstergröße entspricht. Der Durchschnittswert an übertragenen Werten liegt sogar bei unter 2% der Fenstergröße. Die Häufigkeit der Experimente mit nur einem übertragenen Wert erhöhte sich im Vergleich zu $bias = 0$ auf ca. 25%. Erwartungsgemäß verschlechtern sich die Ergebnisse für das tendenziell fallende Signal ($bias = -0{,}2$). Die Ursache dafür ist, dass durch das fallende Signal wesentlich weniger Werte durch andere Werte dominiert werden (vergleiche auch 7.14a). Daraus resultiert, dass durchschnittlich mehr als 65 Werte für den Transfer ausgewählt werden, was sogar den Maximalwert des vorherigen Experiments deutlich übersteigt. Ebenso vergrößert sich die Anzahl für den ungünstigsten Fall, die nun bei etwas mehr als der halben Fenstergröße liegt. Mit nur einem Transferwert kamen nur drei Durchführungen aus.

Führt man die Experimente mit $bias = \{0{,}2; 0; -0{,}2\}$ für weitere Fenstergrößen fort, kann gezeigt werden, wie effizient SemS auch für große Fenster ist und wie die Effizienz von den Eigenschaften des Eingangssignals abhängt. Die Ergebnisse dieser Untersuchung werden nachfolgend im Einzelnen diskutiert.

(a) Histogramm für 100000 Experimente mit $w_o = w_n = 15$ und $bias = 0$



(b) Histogramm für 100000 Experimente mit $w_o = w_n = 250$ und $bias = 0$

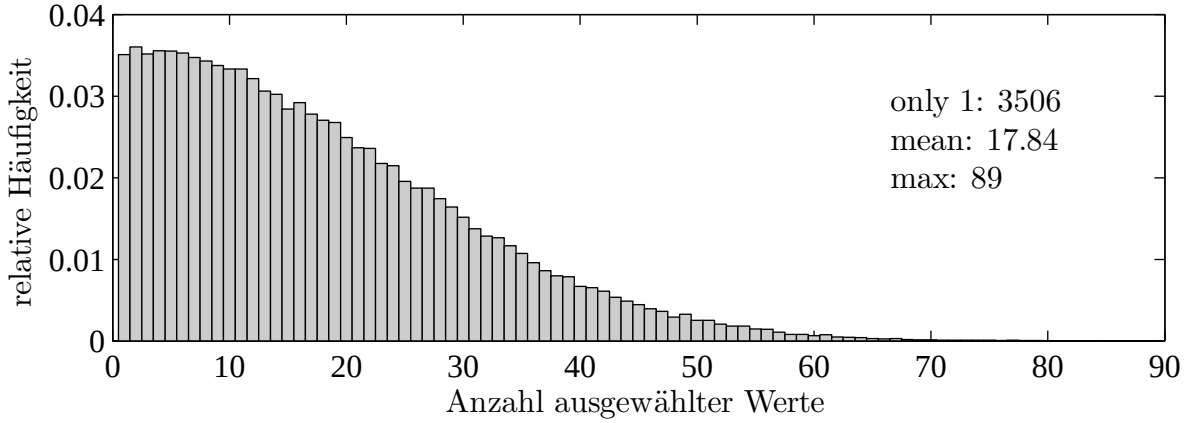
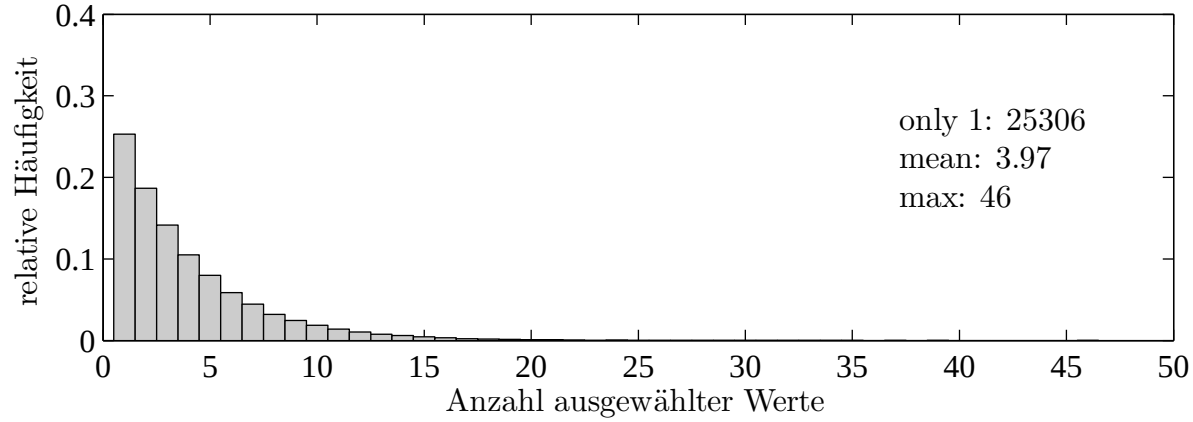


Abbildung 7.16: Untersuchung der Anzahl der ausgewählten Werte für SemS bei unterschiedlichen Fenstergrößen eines Maximumoperators

(a) Histogramm für 100000 Experimente mit $w_o = w_n = 250$ und $bias = 0.2$



(b) Histogramm für 100000 Experimente mit $w_o = w_n = 250$ und $bias = -0.2$

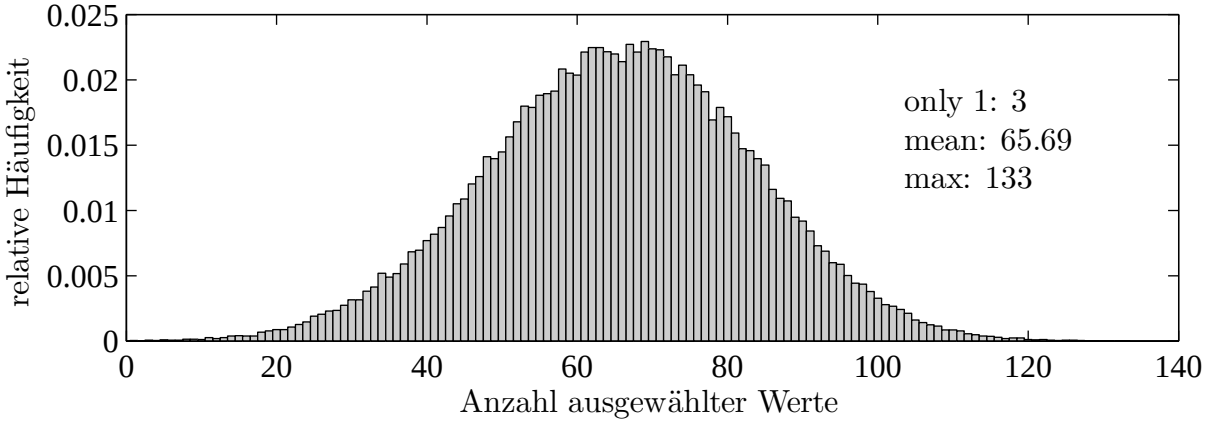


Abbildung 7.17: Untersuchung der Anzahl der ausgewählten Werte für SemS (Maximumoperator) bei tendenziell ansteigenden und abfallenden Sensorsignalen

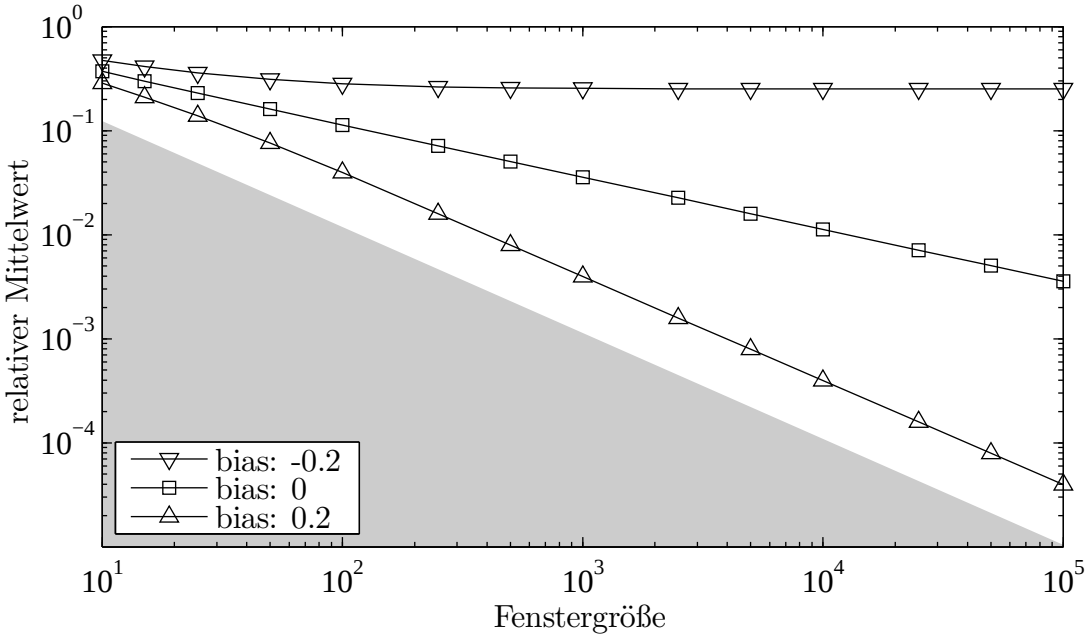


Abbildung 7.18: Durchschnittliche Anzahl ausgewählter Werte mit SemS (normiert)

Abbildung 7.18 zeigt den relativen Mittelwert für die drei Signaltypen und für Fenstergrößen zwischen 10 und 100000 Werten in einem Diagramm mit doppelt logarithmischer Darstellung. Jedem dargestellten Punkt liegen wieder jeweils 100000 Durchläufe zugrunde. Die graue Fläche markiert dabei den Bereich, in dem keine Ergebnisse liegen können, da immer mindestens der neueste Wert aus der Transfermenge ausgewählt wird und die untere Grenze somit bei $1/(w-1)$ liegt. Die Oberkante des Diagramms repräsentiert den ungünstigsten Fall, d. h. alle Werte der Transfermenge wurden ausgewählt. Daraus folgt, ein Migrationsergebnis ist umso besser, je näher es am grauen Bereich liegt.

Wie bei dem zuvor dargestellten Einzelfall, können die besten Ergebnisse für tendenziell steigende Signale erreicht werden. Das bedeutet konkret, für Signale mit $bias = 0{,}2$ sinkt die durchschnittliche Anzahl übertragener Werte von etwa 29% bei $w = 10$ auf 0,004% bei $w = 100000$. Es ist zu beobachten, dass ab einer Fenstergröße von 100 Werten durchschnittlich immer vier Werte übertragen werden. Für Signale ohne *bias* reduziert sich der Anteil von 37% bei $w = 10$ auf 0,35% bei $w = 100000$. Die absolute Anzahl erhöht sich dabei kontinuierlich mit der Fenstergröße. Die tendenziell fallenden Signale stellen jeweils den ungünstigsten Fall dar. Für die ausgewählte Konfiguration mit $bias = -0{,}2$ wird durchschnittlich fast die Hälfte aller Werte bei $w = 10$ übertragen. Der Wert pendelt sich ab einer Fenstergröße von 500 Werten bei ca. 25% ein.

In Abbildung 7.19 ist der relative Maximalwert für die gleiche Gruppe von Experimenten dargestellt. Es sei nochmals erwähnt, dass diese Werte zwar die ungünstigsten Fälle repräsentieren, diese jedoch äußerst selten auftreten (vgl. Histogramme). Für die Fenster mit einer Größe von 10 und 20 Werten gibt es für alle drei Signale Fälle, in

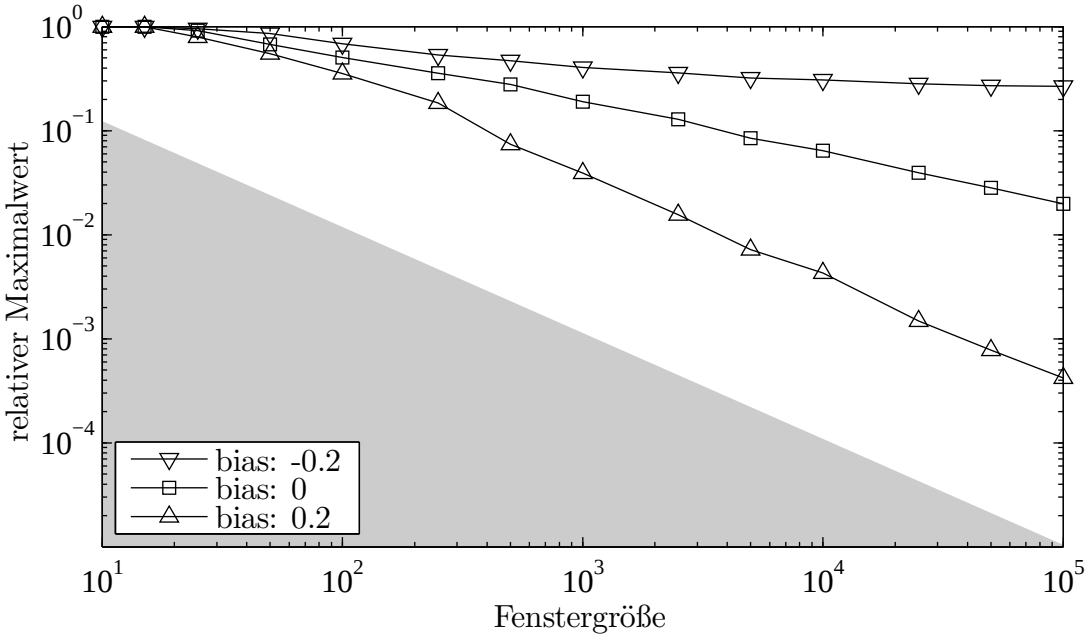


Abbildung 7.19: Maximale Anzahl ausgewählter Werte mit SemS (normiert)

denen alle Werte der Transfermenge übertragen wurden. Bereits ab einer Fenstergröße von $w = 50$ wurde niemals die vollständige Transfermenge übertragen. Für alle Signale ist mit wachsender Fenstergröße ein monoton fallendes Verhalten zu beobachten. Beim größten Fenster (100000 Werte) erreicht das tendenziell steigende Signal einen relativen Maximalwert von 0,04%. Das Signal mit $bias = 0$ liegt immerhin bei unter 2%. Das tendenziell fallende Signal hat mit 26,6% wieder den höchsten und somit ungünstigsten Wert. Dieser liegt zudem vergleichsweise nah am relativen Mittelwert für diesen Signaltyp.

Die relative Häufigkeit für den Fall „only 1“ ist in Abbildung 7.20 dargestellt. Am häufigsten konnte der Fall jeweils für das tendenziell steigende Signal erreicht werden. Für das kleinste Fenster (10 Werte) liegt der Anteil bei fast einem Drittel aller Versuche. Der Wert sinkt anfangs mit steigender Fenstergröße, nähert sich aber bereits ab $w = 50$ einem Wert von ca. 25% an, d. h. etwa ein Viertel aller Versuche konnte mit nur einem Transferwert abgeschlossen werden. Die Beobachtung des Signals ohne *bias* zeigt bei der Anzahl der Versuche und somit auch bei der relativen Häufigkeit ein monoton fallendes Verhalten beginnend bei rund 20% für das kleinste untersuchte Fenster und endend bei etwa 0,15% für das größte. Eine Annäherung an einen bestimmten Wert konnte für die untersuchten Fenstergrößen nicht beobachtet werden. Für das tendenziell fallende Signal konnten nur bis zu einer Fenstergröße von 250 Werten Versuche mit nur einem Transferwert gezählt werden. Der Anteil liegt bereits für die kleinste Fenstergröße bei unter 10%. Bei $w = 250$ wurden nur drei entsprechende Versuche beobachtet.

Aus diesen Untersuchungen lässt sich schlussfolgern, dass die Zustandstransferstrategie

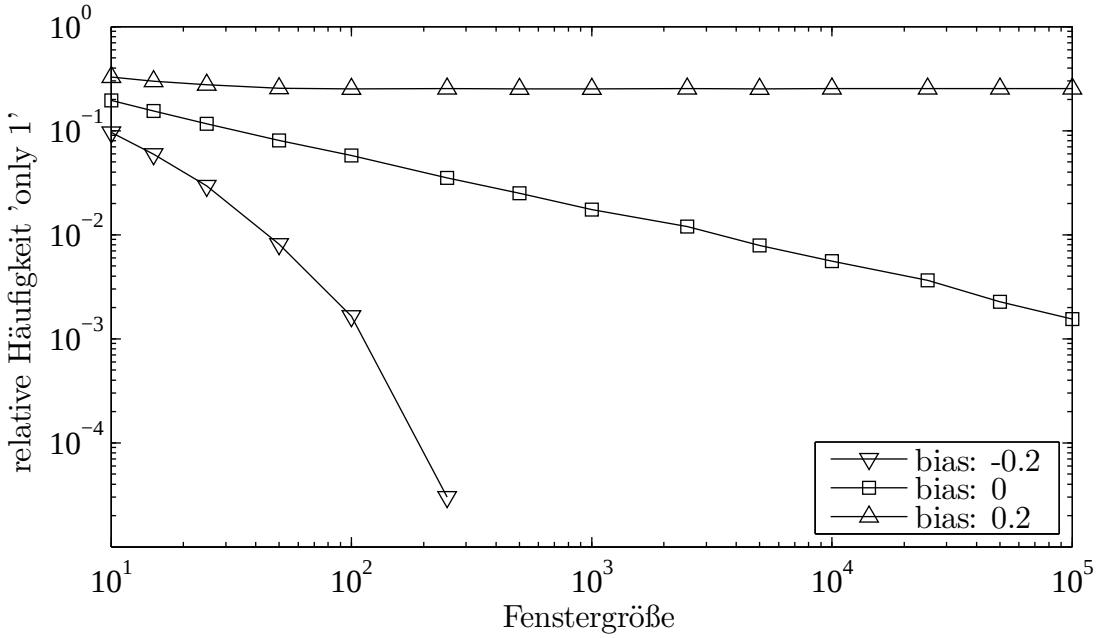


Abbildung 7.20: Anteil der Experimente mit nur einem ausgewählten Wert mit SemS

SemS, angewendet auf Minimum- oder Maximumoperatoren, zur deutlichen Reduzierung der Transfermenge beitragen kann, insbesondere bei großen Fenstern. Selbst bei der ungünstigsten Situation (tendenziell fallendes Signal) liegt der durchschnittliche Anteil ausgewählter Werte unter 50%. Zwar wird sich dieser Anteil mit kleineren Werten für *bias* weiter erhöhen, dies wirkt sich aber lediglich auf die Länge des Zustandstransfers aus. Da durch die Verwendung von SemS jedoch die Möglichkeit besteht, bereits nach dem ersten Transferzyklus zum neuen Operator umzuschalten, stellt eine Verlängerung der Transferzeit kein wesentliches Problem dar.

Dies gilt ebenso für die Migration anderer Aggregationsoperatoren. Für RMS oder einen Operator zur Berechnung eines gleitenden Mittelwertes muss zwar die vollständige Transfermenge übertragen werden, das Umschalten ist aber ebenso nach einem Transferzyklus realisierbar. Experimente mit diesen Operatortypen haben dies bestätigt.

Neben den einfachen Aggregationsanfragen wurde die Beispielanfrage CF für die Evaluierung von SemS genutzt. Dabei wurde wie in Abbildung 7.5 gezeigt, der Zustand aus dem Maximumoperator in den neuen CF-Operator übertragen. Im ersten Transferzyklus wird zunächst die Summe der Quadrate aller Zustandswerte berechnet und übertragen. Anschließend werden wenigstens der älteste und der größte Wert des Zustands transferiert. Mit diesen drei Werten und dem nächsten Eingangswert lässt sich der Crest-Faktor im neuen Operator berechnen. Der Zustandstransfer wird dann mit den ältesten Maximalwerten fortgeführt, bis die Transfermenge vollständig übertragen ist. Auch für diese Anfrage ist durch die Verwendung von SemS ein Umschalten nach dem ersten Transferzyklus möglich.

Prinzipiell lässt sich SemS auch für Fenstervergrößerungen nutzen. Da jedoch weitere Eingangstupel zum Erreichen der neuen Fenstergröße notwendig sind, kommen die Vorteile der Strategie gegebenenfalls nicht zum Tragen. Für alle anderen Fälle mit $w_n \leq w_o$ weist SemS, sofern die Strategie angewendet werden kann, hervorragende Migrationseigenschaften auf. Da die Zustandstransferstrategie SemS sehr anwendungsspezifisch ist und sich von den anderen Strategien wesentlich unterscheidet, wird sie in den nachfolgenden Abschnitten nicht einbezogen. Ihre Evaluierung ist damit abgeschlossen.

Zusammenfassung Durch die Untersuchungen konnte eine korrekte Funktionsweise für alle Zustandstransferstrategien festgestellt werden. Die Strategie SemS ist, sofern sie sich auf eine Anfrage anwenden und mit vertretbarem Aufwand in einem System realisieren lässt, aufgrund ihrer Migrationseigenschaften zu bevorzugen. Für allgemeine Anfragen weisen die Strategien OF, LF+, RSD+ und DS+ gleiche Migrationseigenschaften auf und unterscheiden sich lediglich in der Übertragungsreihenfolge der Zustandswerte. Für die Strategien LF, RSD und DS kann unter Umständen kein vollständiger Transfer garantiert werden, was dann zu einer schlechteren Migrationsdauer im Vergleich zu den anderen Migrationsstrategien führen kann. Die Untersuchungsergebnisse in den nächsten Abschnitten werden die genauen Randbedingungen dafür veranschaulichen. Funktional zwar korrekt, weist die Strategie RS die schlechtesten Migrationseigenschaften auf, weshalb sie in den folgenden Abschnitten nicht berücksichtigt wird.

7.5 Vergleich zu existierenden Lösungen

In diesem Abschnitt wird das Migrationsverhalten unter Anwendung des numerischen und des heuristischen Verfahrens betrachtet. So erfolgt die Evaluierung des numerischen Verfahrens in periodischen und schwankungsbeschränkten Datenströmen. Für das heuristische Verfahren werden zudem aperiodische Datenströme einbezogen. Dabei wird auf die Migrationseigenschaften der verschiedenen Zustandstransferstrategien im Vergleich zu den existierenden Migrationsstrategien eingegangen. Zunächst erfolgt eine Auswahl geeigneter Migrationsstrategien für den Vergleich. Die Versuchsergebnisse für die verschiedenen Verfahren und Datenstromtypen sowie deren Diskussion folgen im Anschluss.

Für die Ermittlung der Performance-Werte wurde u. a. eine Simulation verwendet. Diese konzentriert sich im Wesentlichen auf die Verarbeitungszeitpunkte der Datentupel, d. h. der neuen Eingangswerte aber auch der Zustandstransferwerte. Das Simulationsmodell lässt sich je nach Datenstromtyp konfigurieren. So sind die Periodendauer (T) bzw. der mittlere Ereignisabstand ($\overline{T}$) und die Anzahl der erzeugten Datenstromelemente pro Periode (N_i) anzugeben. Des Weiteren sind System- und Anfrageeigenschaften zu spezifizieren, z. B. die Verarbeitungsdauer (T_p), die Tupeltransferdauer (T_t) oder die Fenstergrößen (w_o, w_n). Die Auswahl einer Migrationsstrategie und gegebenenfalls die Angabe, wie oft ein Experiment zu wiederholen ist, vervollständigen die Konfiguration. Zur Laufzeit werden dann die Datenströme entsprechend der Konfiguration erzeugt und

eine Migration mit der ausgewählten Strategie wird betrachtet.

7.5.1 Auswahl vergleichbarer Migrationsstrategien

Alle analysierten existierenden Migrationsstrategien wurden in Tabelle 3.2 (S. 58, obere Gruppe) aufgelistet. Das Fazit der Analyse war, dass nur die verallgemeinerte Strategie GPT für einige[12] Migrationsszenarien in einer verteilten Umgebung geeignet ist. GMS und GHM können in verteilten Umgebungen verwendet werden, wenn geringfügige Anpassungen an den Strategien durchgeführt werden. Alle anderen Strategien sind ungeeignet, da sie beispielsweise gemeinsame Zustände oder gemeinsame Warteschlangen verwenden. Für den Vergleich wurden deshalb die Migrationsstrategien GMS, GPT und GHM ausgewählt. Ihr Ablauf wird im Anschluss jeweils kurz wiederholt. Zusätzlich werden eventuelle Anpassungen beschrieben. Die Liste wird vervollständigt durch die neuen Zustandstransferstrategien. Es sei bemerkt, dass die für den Vergleich ausgewählten existierenden Migrationsstrategien über die für sie beschriebenen Anwendungsfälle und Eigenschaften hinaus eingesetzt werden, z. B. verteilte Umgebung, Fensterverkleinerungen, Fenstervergrößerungen oder anzahlbasierte Fenster.

GMS Zur Durchführung der Migration mit GMS wird als erstes die alte Anfrage angehalten und alle Datentupel werden aus den Zuständen der Operatoren der alten Anfrage extrahiert. Diese werden in ihrer ursprünglichen zeitlichen Reihenfolge in die Eingangswarteschlangen der neuen Anfrage eingefügt. Indem diese Werte verarbeitet werden, werden die Zustände in der neuen Anfrage rekonstruiert. Die dabei entstehenden Tupel am Ausgang der neuen Anfrage werden ignoriert. Die Migration endet, wenn alle Eingangswarteschlangen gleichzeitig leer sind. Anschließend werden die inzwischen empfangenen regulären Eingangswerte verarbeitet.

Zur Realisierung von GMS in einer verteilten Umgebung und zum Erreichen von vergleichbaren Ergebnissen wurden folgenden Anpassungen durchgeführt. Während der Migration wird bei Operatoren, welche direkt mit dem Eingangsdatenstrom verbunden sind, eine zusätzliche Eingangswarteschlange verwendet (vgl. Abbildung 5.14), um die transferierten von den regulären Datentupeln zu trennen. Somit können die regulären Eingangswerte während der Initialisierung gepuffert werden. Die Warteschlangen mit den Zustandstransferwerten werden im Unterschied zu den neuen Transferstrategien mit einer höheren Priorität behandelt. Falls der Zustandstransfer auch für Zwischenzustände durchgeführt wird, z. B. der Zustand $\mathcal{Z}_{AB}$ in Abbildung 5.7 (S. 92), ist für die Zwischenzustände die reguläre Warteschlange zu priorisieren, um die Verarbeitung nicht zu blockieren.

In [YKPS07] wird die Migrationsdauer aus der Anzahl der transferierten Datentupel und ihrer Verarbeitungszeit berechnet. Da der Zustandstransfer lokal im Arbeitsspeicher

[12] Nicht für beliebige Migrationsstrategien geeignet, z. B. können wachsende Fenster nicht mittels GPT migriert werden.

erfolgt, wird eine Zustandstransferzeit nicht berücksichtigt. Bei einer Übertragung der Zustandswerte über ein Netzwerk ist die Transferzeit erheblich höher, weshalb sie für die Migrationsdauer berücksichtigt werden muss. Aus diesem Grund wird T_{ZT} in T_M einbezogen.

Ebenso wird in die Migrationsdauer die Zeit eingerechnet, die für die Verarbeitung der angestauten regulären Datentupel notwendig ist. Nur so lässt sich GMS mit anderen Strategien vergleichen. Die Migrationsdauer wird also zwischen dem ersten Wert der Migration und dem ersten Ergebnis, das nicht verzögert ausgegeben wurde, bestimmt. Die genaue Festlegung des Migrationsendes erfolgt dabei entsprechend dem Datenstrom nach den oben beschriebenen Regeln (Abschnitt 7.3).

GPT Zu Beginn der Migration werden alle Eingangsdatenströme mit der neuen Anfrage verbunden. Diese wird dann parallel zur alten Anfrage ausgeführt, wobei die Ergebnisausgabe während der Migration durch die alte Anfrage erfolgt. Ergebnisse der neuen Anfrage werden ignoriert, da die Migrationsdauer genutzt wird, um die inneren Zustände der Anfrage zu füllen. Nach Abschluss dieses Vorgangs wird zur neuen Anfrage umgeschaltet. Die Strategie GPT kann ohne Anpassungen verwendet werden.

GHM Die Migrationsstrategie GHM ist eine Kombination von GMS und GPT. Die Migration beginnt mit dem Verbinden der Eingangsdatenströme mit FIFO-Warteschlangen an den Eingängen der neuen Anfrage. Danach werden die Zustände aus der alten Anfrage vollständig extrahiert und in ihrer zeitlichen Reihenfolge in die Warteschlangen eingefügt, um die neue Anfrage damit zu initialisieren. Während der Migration werden beide Anfragen parallel ausgeführt, wobei die alte Anfrage für die Ergebnisausgabe zuständig ist und die Initialisierung der neuen Anfrage während der Pausenzeit des Systems durchgeführt wird. Das Umschalten zur neuen Anfrage nach Abschluss der Migration erfolgt, wenn alle Eingangswarteschlangen zum ersten Mal gleichzeitig leer sind.

In dieser Form kann GHM nicht in einer verteilten Umgebung genutzt werden. Deshalb wurde die Strategie wie folgt angepasst. Bei der in [YKPS07] beschriebenen zentralisierten Ausführung wird davon ausgegangen, dass die Transferwerte hinreichend schnell in die Warteschlangen eingefügt werden, ohne mit dem regulären Ablauf zu kollidieren. In verteilten Umgebungen ist dies nicht möglich. Deshalb wird während der Migration neben der Warteschlange für die regulären Werte eine zusätzliche Warteschlange nur für Transferwerte genutzt. Der Zustandstransfer wird dann während den Pausenzeiten durchgeführt. Um Kollisionen mit den Eingangswerten zu vermeiden, wird die gleiche Art der Konfiguration wie bei den neuen Transferstrategien verwendet, z. B. die Anzahl der Transferwerte pro Pause oder der Beginn des Zustandstransfers. Dieser wird nach dem ersten Eingangswert gestartet. Mit GHM wird der Zustand vollständig übertragen. Aufgrund der vorherigen Anpassung reduziert sich diese Menge bei der Verwendung des numerischen Verfahrens ohne sofortigen Zustandstransfer auf $w - 1$. Da mit GHM die Tupel im neuen Operator in ihrer zeitlichen Reihenfolge verarbeitet werden, werden

während des Zustandstransfers alle neuen Eingangswerte in ihrer jeweiligen Eingangswarteschlange gepuffert. Ihre Verarbeitung wird nach Abschluss des Zustandstransfers durchgeführt.

Die Bestimmung des Migrationsendes anhand geleerter Warteschlangen ist für manche Migrationsszenarien nicht ausreichend, z.B. bei Fenstervergrößerung. Deshalb werden zusätzlich die Zustände der neuen Anfrage überwacht. Die Migration endet, wenn alle Fenster vollständig gefüllt sind.

Neue Zustandstransferstrategien Die neuen Transferstrategien lassen sich aufgrund ihrer Migrationseigenschaften gruppieren, was durch die nachfolgend dokumentierten Experimente bestätigt wird. Dabei bilden OF, LF+, RSD+, DS+ die Gruppe $\mathcal{G}_{OF}$. Bei korrekter Konfiguration und geeignetem Szenario, z. B. bei den Beispielanfragen RMS und CF, kann auch DS dazu gezählt werden. Die Strategien LF und RSD gehören nicht zu dieser Gruppe, da deren Konfiguration je nach Anwendungsfall zu abweichenden Ergebnissen führt. Sollten jedoch alle Migrationsstrategien in einem Anwendungsfall die gleichen Ergebnisse produzieren, werden sie zu einer Gruppe $\mathcal{G}$ zusammengefasst, d. h. $\mathcal{G} = \{\mathcal{G}_{OF}, \text{LF}, \text{RSD}\}$. Die Ursachen für gleiche oder unterschiedliche Ergebnisse werden an entsprechender Stelle erläutert. Die Strategie RSD stellt einen Sonderfall dar, da ihre Ergebnisse immer zwischen den Resultaten von $\mathcal{G}_{OF}$ und LF liegen. Die Ursache dafür ist die zufällige Auswahl der Transferwerte. Dies kann dazu führen, dass Zustandswerte verworfen werden, bevor sie übertragen werden konnten. Deshalb kann gegebenenfalls kein eindeutiges Ergebnis für diese Strategie ermittelt werden, weshalb auf eine Angabe von Ergebnissen verzichtet wird. Das gleiche gilt für die Strategien RS und SemS. Die Gründe wurden bereits diskutiert.

7.5.2 Numerisches Verfahren

Die Evaluierung des numerischen Verfahrens (Kapitel 5) wurde anhand der Beispielanfragen RMS und CF sowohl im OSGi-Prototyp als auch simulativ durchgeführt. Es wurden verschiedene Konfigurationen untersucht, von denen eine für die Dokumentation der Ergebnisse ausgewählt wurde. Die Konfigurationsunterschiede bestanden dabei in den Eigenschaften der Eingangsdatenströme, z. B. Periodendauer oder Schwankungen, in der Länge der gewählten Verarbeitungs- und Transferzeiten sowie in den Fenstergrößen der Original- und Zieloperatoren. Zwar unterscheiden sich die Ergebnisse der Experimente quantitativ je nach Konfiguration, prinzipiell sind sich die Ergebnisse jedoch ähnlich.

Der Performance-Vergleich der Zustandstransfermethoden wurde sowohl für periodische als auch für schwankungsbeschränkte Eingangsdatenströme durchgeführt. Aufgrund der Tatsache, dass es sich im Prototyp generell um schwankungsbehaftete Datenströme handelt, lassen sich periodische (schwankungsfreie) Ströme nur per Simulation erzeugen. Dokumentiert sind die Ergebnisse der Simulation mit periodischen Eingangsströmen, da diese den idealen Migrationsverlauf der jeweiligen Strategie repräsentieren und deren

Korrektheit somit einfach nachzuvollziehen ist. Die Ergebnisse der periodischen Datenströme dienen außerdem als Referenz für nachfolgende Experimente, um diese nicht nur qualitativ sondern auch quantitativ vergleichen zu können. Aus diesem Grund werden die periodischen Datenströme genau wie schwankungsbeschränkte Datenströme konfiguriert, wenngleich die Möglichkeit bestünde, mehr Zustandswerte während einer Pause zu übertragen.

Der Vergleich von schwankungsbeschränkten mit periodischen Datenströmen zeigt folgendes. In Bezug auf die Migrationsdauer (T_M) liegen die einzelnen Ergebnisse der schwankungsbeschränkten Datenströme maximal $\tau+\tau'$ von den Werten der periodischen Ströme entfernt, da der erste und der letzte reguläre Wert der Migration über die Abweichung entscheiden. Die Mittelwerte der Migrationsdauer in schwankungsbeschränkten Strömen mit $\tau = \tau'$ entsprechen den Werten der periodischen Datenströme. Eine Abweichung bezüglich der Anzahl der übertragenen Werte (N_o) bei gleicher Konfiguration gibt es nicht.

Nachfolgend sind die Ergebnisse für vier verschiedene Experimente aufgeführt. Gleichbleibende Experimentbedingungen sind:

- $T = 100\,\text{ms}$
- $N_i = 1$
- $T_p = 2\,\text{ms}$
- $T_t = 10\,\text{ms}$
- $w_o = 2500$

Variiert wurde die Größe des neuen Fensters:

- $w_n = 2500$ (gleichgroße Fenster)
- $w_n = 2000$ (Fensterverkleinerung)
- $w_n = 2600$ (Fenstervergrößerung)
- $w_n = 3000$ (Fenstervergrößerung)

Zudem gelten für die einzelnen Migrationsstrategien folgende Bedingungen. Bei GMS ist für jedes übertragene Tupel eine Verarbeitungszeit zu berücksichtigen egal ob Nachfolgeoperatoren existieren oder nicht. Da sich das Fenster kontinuierlich füllt, steigt T_p bis auf 2 ms beim vollständig gefüllten Fenster an. In der Simulation wird daher für Zustandstransferwerte eine mittlere Verarbeitungsdauer von 1ms verwendet.

Bei GHM tritt wie bei GMS für jedes übertragene Tupel eine Verarbeitungszeit auf. Um eine kollisionsfreie Migration zu gewährleisten, ist der Zustandstransfer rechtzeitig zu beenden, was durch die Einplanung einer entsprechenden Reserve r (vgl. Gleichung 5.10) realisiert wird – im Prototyp 15 ms aufgrund des Verhaltens der JVM[13],

[13] vgl. Abschnitt 7.1.3

d. h. $r = 15\,\text{ms}/T$. Abzüglich der Verarbeitungszeit für ein reguläres Datentupel verbleibt eine Zeit von 83 ms was bei GHM für den Transfer von sieben Zustandswerten ($N_t = 7$, Gesamttransferdauer pro Pause 77 ms) ausreicht[14].

Die Verwendung der neuen Transferstrategien erlaubt die Unterdrückung der Verarbeitung im Zieloperator, sodass lediglich der Zustand dieses Operators gefüllt wird. Da bei den Beispielanfragen keine nachfolgenden Operatoren zu initialisieren sind, kann also auf eine Verarbeitung der Zustandstransferwerte verzichtet werden. Deshalb können während einer Pause acht Zustandstupel übertragen werden ($N_t = 8$, Gesamttransferdauer pro Pause 80 ms).

Für den Vergleich der verschiedenen Transferstrategien werden die in Abschnitt 7.3 beschriebenen Größen herangezogen: Anzahl übertragener Zustandswert (N_o), Zustandstransferdauer (T_{ZT} in s), Migrationsdauer (T_M in s) sowie die Ergebnisverzögerung (T_{OS} in s). Für Fenstervergrößerungen ist zudem die Anzahl der zusätzlichen neuen Werte angegeben, die benötigt werden, um das neue Fenster nach dem Zustandstransfer auf die Zielgröße zu vervollständigen.

Gleichgroße Fenster Die Ergebnisse für eine Migration zwischen gleichgroßen Fenstern ($w_o = w_n = 2500$) sind in Tabelle 7.1 dokumentiert. Die Tabelle wird ergänzt durch eine Spalte *ideal*, welche die Performance-Ziele der Migration symbolisiert, d. h. die Migrationsdauer soll so gering wie möglich sein und gleichzeitig darf es zu keiner Ergebnisverzögerung kommen. Die Art und Weise, wie diese Ziele erreicht werden ist dabei nicht relevant, weshalb für N_o und T_{ZT} keine Werte angegeben sind.

Im konkreten Fall sind die Ergebnisse aller neuen Transferstrategien (OF, LF, LF+, RSD, RSD+, DS, DS+) identisch, weshalb sie in der Gruppe $\mathcal{G}$ zusammengefasst sind. Sie erreichen jeweils die kürzeste Migrationsdauer im Vergleich zu den existierenden Strategien. Dies liegt daran, dass weniger Werte übertragen werden als bei GMS und GHM. GPT benötigt aufgrund des Parallelverfahrens immer deutlich länger, weshalb auch bei den nachfolgenden Konfigurationen nicht näher darauf eingegangen wird. Bei GMS fällt auf, dass für einen erheblichen Teil der Migration (27,5 s) keine Ergebnisse ausgegeben werden. Dieses Verhalten dauert auch über die Zustandstransferzeit an, da direkt nach dem Transfer zunächst die transferierten Werte verarbeitet werden, die Ergebnisse aber verworfen werden. Danach beginnt die Verarbeitung der währenddessen empfangenen Eingangswerte, welche mit einer maximalen Verzögerung von 27,5 s ausgegeben werden. Erst nach dem Migrationsende erfolgt die Ergebnisausgabe verzögerungsfrei.

Eine Ergebnisausgabe ohne Verzögerung ist auch bei GHM gegeben. Hier dauert die Migration im Vergleich zu den neuen Transferstrategien etwa 31% länger. Ursache ist zum einen die Anzahl der Tupel, die während einer Pause übertragen werden können (7 bei GHM, 8 bei $\mathcal{G}$). Zum anderen ist bei GHM eine Beschränkung der Transferwerte auf notwendige Werte nicht vorgesehen, weshalb der gesamte Zustand des Originalopera-

[14] Die Ergebnisse für GHM mit $N_t = 8$ (unter Vernachlässigung der Reserve) werden in der Zusammenfassung dieses Abschnitts diskutiert.

tors übertragen werden muss. Dies erfordert zusätzliche Transferzeit und führt zu einer längeren Migrationsdauer.

Migrationsstrategie	$\mathcal{G}$	GMS	GPT	GHM	*ideal*
übertragene Werte (N_o)	2221	2500	0	2499	
Zustandstransferdauer (T_{ZT} in s)	27,75	25,00	-	35,67	
Migrationsdauer (T_M in s)	27,80	28,06	249,90	36,70	minimal
Ergebnisverzögerung (T_{OS} in s)	0,00	27,50	0,00	0,00	0,00

Tabelle 7.1: Vergleich verschiedener Migrationsstrategien bei einer Migration zwischen gleichgroßen Fenstern ($w_o = w_n = 2500$)

Fensterverkleinerung Bei einer Fensterverkleinerung ($w_o = 2500$, $w_n = 2000$) fällt der zuletzt beschriebene Effekt besonders ins Gewicht. Während sich bei allen neuen Transferstrategien die Anzahl der Transferwerte und somit auch die Migrationsdauer reduziert, ist bei GMS und GHM keine Veränderung zu beobachten (Tabelle 7.2). Die Migrationsdauer für GMS und GHM liegt dadurch ca. 26% bzw. 63% über der von $\mathcal{G}$. Lediglich die Migrationsdauer mit GPT verringert sich proportional zur neuen Fenstergröße, der absolute Wert liegt jedoch immer noch weit über den Ergebnissen aller anderen Strategien.

Migrationsstrategie	$\mathcal{G}$	GMS	GPT	GHM
übertragene Werte (N_o)	1776	2500	0	2499
Zustandstransferdauer (T_{ZT} in s)	22,18	25,00	-	35,67
Migrationsdauer (T_M in s)	22,30	28,06	199,90	36,70
Ergebnisverzögerung (T_{OS} in s)	0,00	27,50	0,00	0,00

Tabelle 7.2: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fensterverkleinerung ($w_o = 2500$ und $w_n = 2000$)

Fenstervergrößerung Für Fenstervergrößerungen sind nachfolgend zwei unterschiedliche Beispiele dargestellt. Beiden gemeinsam ist, dass die Ergebnisse von LF und RSD von den Ergebnissen der anderen neuen Transferstrategien abweichen. Aus diesem Grund ist LF separat aufgeführt. Die Gruppe $\mathcal{G}_{OF}$ vereint die Strategien OF, LF+, RSD+, DS, DS+. Wie bereits erläutert, liegen die Ergebnisse von RSD jeweils zwischen denen von $\mathcal{G}_{OF}$ und LF.

Im ersten Beispiel (Tabelle 7.3) beträgt die neue Fenstergröße $w_n = 2600$. Die Ergebnisse für GMS und GHM entsprechen wieder dem ersten Fall. Insgesamt wird die

schnellste Migration nun von GMS erreicht, jedoch bleibt der Nachteil der Ergebnisverzögerung bestehen. Vergleicht man die Strategien ohne Ergebnisverzögerung, erreicht $\mathcal{G}_{OF}$ das beste Resultat, die Migrationsdauer liegt mindestens 25% unter der von GHM oder LF. Die Ursache für die längere Migration mit LF ist, dass Werte aus dem Originaloperator verworfen werden, bevor sie transferiert werden können. Deshalb ist für LF mit der vorgegebenen Konfiguration bei $N_o = 2222$ die maximale Anzahl übertragbarer Werte erreicht. Für die Strategien der Gruppe $\mathcal{G}_{OF}$ liegt dieser Wert unter gleichen Bedingungen bei 2499. Die fehlenden Werte bei Zustandstransferende müssen bei LF im dargestellten Fall mit 99 zusätzlichen neuen Werten kompensiert werden, was die Migrationsdauer beträchtlich verlängert.

Migrationsstrategie	$\mathcal{G}_{OF}$	LF	GMS	GPT	GHM
übertragene Werte (N_o)	2310	2222	2500	0	2499
Zustandstransferdauer (T_{ZT} in s)	28,86	27,76	25,00	-	35,67
Migrationsdauer (T_M in s)	28,90	37,70	28,06	259,90	36,70
Ergebnisverzögerung (T_{OS} in s)	0,00	0,00	27,50	0,00	0,00
zusätzliche neue Werte ($N_{n,add}$)	0	99	0	-	0

Tabelle 7.3: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fenstervergrößerung ($w_o = 2500$ und $w_n = 2600$)

Migrationsstrategie	$\mathcal{G}_{OF}$	LF	GMS	GPT	GHM
übertragene Werte (N_o)	2499	2222	2500	0	2499
Zustandstransferdauer (T_{ZT} in s)	31,23	27,76	25,00	-	35,67
Migrationsdauer (T_M in s)	50,00	77,70	50,00	299,90	50,00
Ergebnisverzögerung (T_{OS} in s)	0,00	0,00	27,50	0,00	0,00
zusätzliche neue Werte ($N_{n,add}$)	187	499	220	-	136

Tabelle 7.4: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fenstervergrößerung ($w_o = 2500$ und $w_n = 3000$)

Im zweiten Beispiel (Tabelle 7.4) beträgt die neue Fenstergröße $w_n = 3000$. Die Migrationsdauer beträgt sowohl für die Strategien der Gruppe $\mathcal{G}_{OF}$ als auch für GMS und GHM 50 s. Da sich bei Migrationsende das neue Fenster zum einen aus neuen Werten und zum anderen aus Zustandstransferwerten zusammensetzt, der Transfer aber schneller endet als die neuen Werte empfangen werden, ist die Zeitspanne zum Empfang neuer Werte für die Migrationsdauer ausschlaggebend. Die Differenz zwischen beiden Zeiten, also die Zeit, in der ausschließlich auf neue Werte zur Vervollständigung des Fensters gewartet wird, lässt sich u. a. von der Anzahl zusätzlicher neuer Werte ableiten. Für

LF ist dafür wieder ein höherer Wert zu beobachten, weshalb die Migration mit LF entsprechend länger dauert.

Zusammenfassung Über die vier beschriebenen Fälle hinaus wurde das numerische Verfahren mit der gegebenen Konfiguration für alle Größen des neuen Fensters zwischen 2 und 6000 evaluiert. Die Entwicklung der Migrationsdauer (T_M) in Abhängigkeit der neuen Fenstergröße (w_n) ist in den Abbildungen 7.21 und 7.22 gezeigt, wobei Abbildung 7.22 einen Ausschnitt im Bereich $2000 \leq w_n \leq 3000$ darstellt. Die neuen Fenstergrößen der vier Fälle sind jeweils mit vertikalen Linien markiert ($w_n = \{2000, 2500, 2600, 3000\}$).

Die Ergebnisse für die neuen Strategien sind bis zum Punkt $w_n = w_o = 2500$ identisch. Danach steigt die Migrationsdauer für LF schneller, weil auf zusätzliche neue Werte gewartet werden muss, um das neue Fenster zu vervollständigen. Bei $\mathcal{G}_{OF}$ tritt dieser Effekt etwas später ein. Bei der verwendeten Konfiguration ist genau bei $w_n = 2814$ der erste zusätzliche Wert nötig. Die Differenz der Migrationsdauer von LF und OF bleibt dann für alle größeren Fenster konstant.

Bei GMS und GHM ist zu sehen, dass die Migrationsdauer bis zu einem gewissen Punkt konstant verläuft. Dabei ist die Migration mit GMS etwas schneller als mit GHM, da bei GMS die Zustandswerte mit Volllast übertragen werden, wohingegen bei GHM der Transfer zugunsten der regulären Werte regelmäßig unterbrochen werden muss. Führt man eine Migration mit GHM unter Vernachlässigung der Reserve mit $N_t = 8$ aus, kann die Migrationsdauer für diesen Bereich um 4,5 s reduziert werden, der Wert von $\mathcal{G}_{OF}$ wird jedoch nicht unterboten. Der konstante Verlauf wird von einem ansteigenden Verhalten abgelöst, wenn die übertragenen Zustandswerte nicht mehr ausreichen, um das neue Fenster zu füllen, und zusätzliche neue Werte notwendig sind. Die Migrationsdauer entspricht dann der von $\mathcal{G}_{OF}$, sieht man von dem Abschnitt ab, in dem GMS schneller als $\mathcal{G}_{OF}$ ist. Die Migrationsdauer von GPT steigt von Anfang an proportional mit der Fenstergröße. In Abbildung 7.22 liegt sie außerhalb des dargestellten Bereichs.

Unter Einhaltung der Anforderung, die Migration ohne negative Beeinflussung des Primärsystems durchzuführen, d. h. unter anderem ohne Ergebnisverzögerung, sind die neuen Transferstrategien in jedem Fall besser geeignet als die existierenden. Ist eine Ergebnisverzögerung akzeptabel, ist GMS für einen geringen Bereich besser. Für Migrationsszenarien zwischen gleichgroßen Fenstern und bei Fensterverkleinerung ist die Transferstrategie LF zu bevorzugen, zumal diese Strategie ohne Berechnung von Transferparametern auskommt[15].

Zum Nachweis der Funktionsfähigkeit der Migrationsstrategien unter Berücksichtigung der Reserve, wurden mehrere Experimente mit verschiedenen Konfigurationen im Prototyp durchgeführt. Dabei wurde u. a. das Verhalten des Ausgangsdatenstroms im

[15] Aus diesem Grund kann LF für $w_n \leq w_o$ zur Validierung der Zustandstransferstrategien OF, LF+, RSD, RSD+, DS, DS+ genutzt werden, für die N_o berechnet werden muss, um den ältesten Wert, der übertragen wird, zu bestimmen. Zur Validierung werden die Mengen der übertragenen Werte der Strategien aus $\mathcal{G}_{OF}$ und von LF miteinander verglichen. Stimmen diese überein, wurde der Wert für N_o korrekt berechnet.

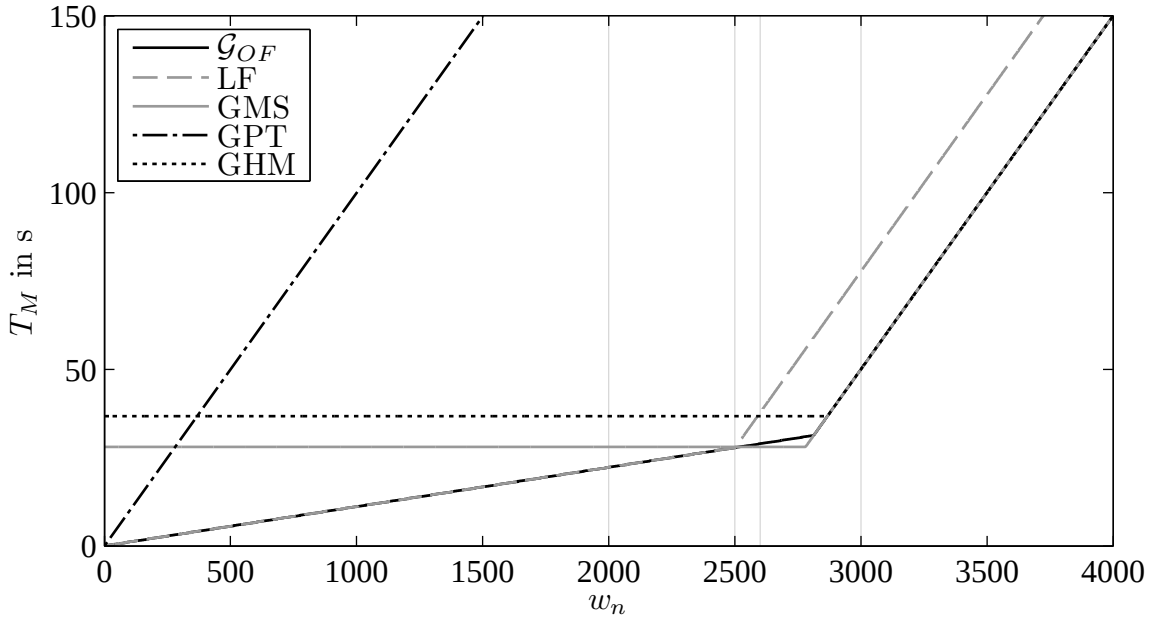


Abbildung 7.21: Migrationsdauer für verschiedene Migrationsstrategien

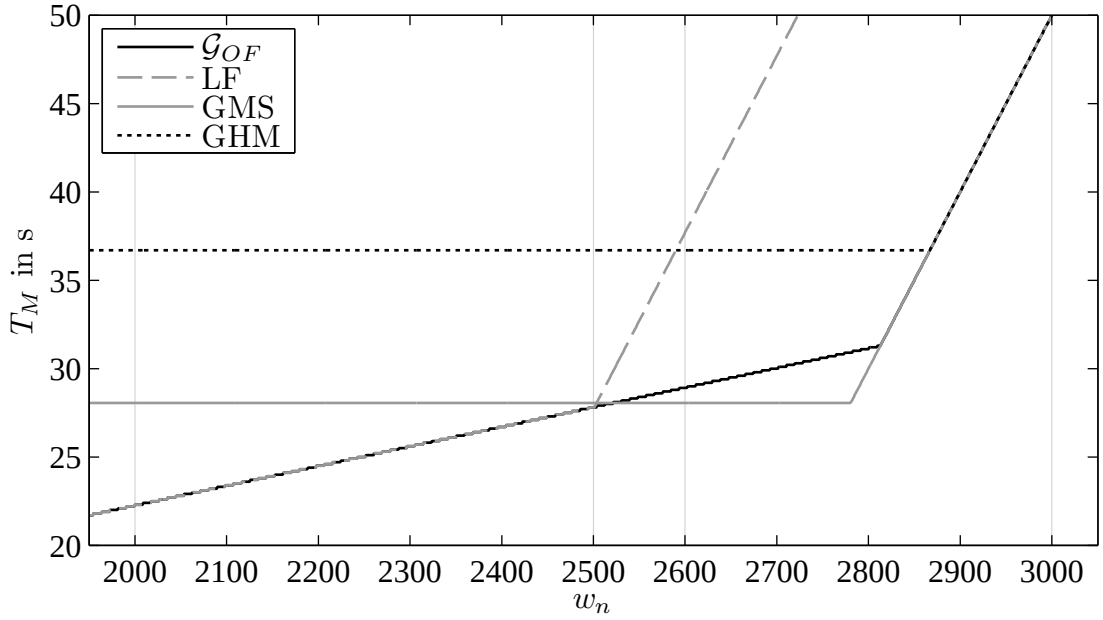


Abbildung 7.22: Migrationsdauer (Ausschnitt) für verschiedene Migrationsstrategien

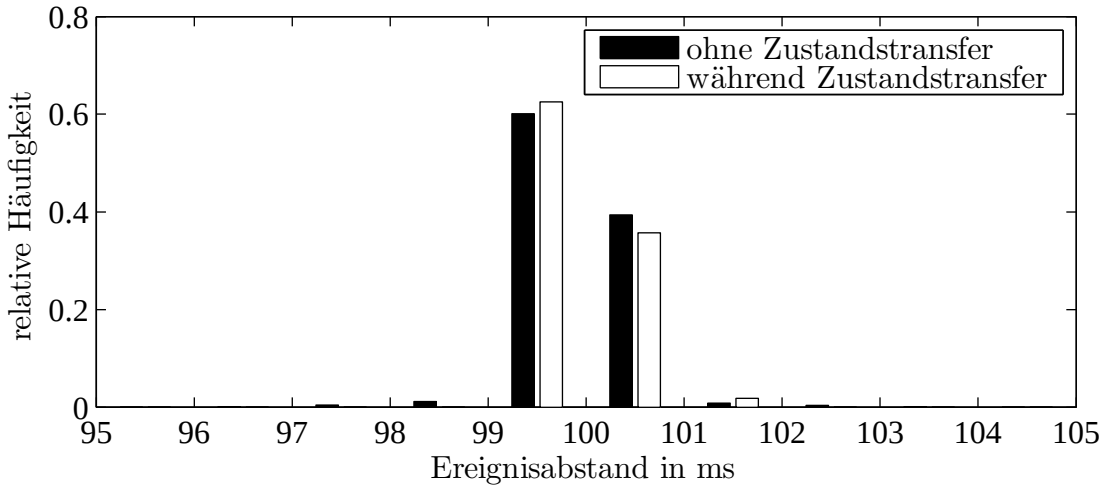


Abbildung 7.23: Einfluss der Migration auf den Ergebnisdatenstrom

regulären Betrieb und während der Migration beobachtet. Ein beispielhaftes Ergebnis ist in Abbildung 7.23 dargestellt. So wurden die Ereignisabstände im Ausgangsdatenstrom ermittelt und ihre relative Häufigkeit im Histogramm gegenübergestellt. Für eine Zeitspanne ohne Zustandstransfer häufen sich die Werte um 100 ms, was der Periodendauer des Eingangsdatenstroms entspricht. Es gibt wenige Werte, die davon abweichen. Diese befinden sich im Allgemeinen zwischen 95 ms und 105 ms und mit abnehmender Häufigkeit, je weiter sie sich von 100 ms entfernen. Es sei wiederholt, dass es, ausgelöst durch die JVM, in äußerst seltenen Fällen im regulären Betrieb zu Abweichungen von bis zu 15 ms kommt.

Während der Migration gibt es im dargestellten Beispiel nur einen Wert, der mehr als 1 ms vom erwarteten Ereignisabstand abweicht. Beim Verhalten des Ausgangsdatenstroms ist also kein Unterschied zwischen Migration und regulärem Verhalten zu beobachten, was eine einwandfreie Funktionsweise des Migrationsverfahrens belegt. Dies gilt sowohl im konkreten Beispiel als auch für alle weiteren durchgeführten Experimente.

Die Experimente wurden im Prototyp ebenso mit der Option „sofortiger Zustandstransfer“ für die neuen Zustandstransferstrategien durchgeführt. Dabei startet der Zustandstransfer direkt bei Migrationsbeginn, ohne auf den ersten regulären Eingangswert zu warten. Dies erfordert jedoch die Überwachung des Datenstroms, um die verbleibende Zeit bis zum ersten regulären Eingangswert zu kennen. Auch für diese Variante muss die Reserve eingehalten werden, um eine Kollision des Zustandstransfers mit der regulären Verarbeitung zu vermeiden. Im besten Fall kann damit genau eine Zeit von einer Periodendauer eingespart werden, in den oberen Beispielen also 0,1 s. Der zusätzliche Aufwand für die vergleichsweise geringe Ausbeute erscheint sehr hoch, zumal eine Ersparnis nicht für jede Konfiguration zu erreichen ist.

7.5.3 Heuristisches Verfahren

Das heuristische Verfahren (Kapitel 6) wurde ebenfalls unter Verwendung des Prototyps und simulativ evaluiert. Dafür wurden basierend auf den Beispielanfragen verschiedene Szenarien erstellt, deren Evaluierungsergebnisse im Anschluss dokumentiert sind. Die Szenarien beinhalten Einzelfenster oder mehrere Fenster kombiniert mit periodischen oder aperiodischen Eingangsdatenströmen.

Für die Evaluierung werden die neuen Zustandstransferstrategien ebenso gruppiert wie bisher. Auch für den Vergleich werden wieder die gleichen existierenden Migrationsstrategien verwendet. Die Bemessung der Migrationsdauer wird für das heuristische Verfahren leicht modifiziert. So kann das Migrationsende hier auch durch den letzten Zustandstransferwert markiert werden. Da im Anschluss an den letzten Zustandstransferwert in aperiodischen Datenströmen eine unbestimmte Zeit auf den nächsten regulären Eingangswert gewartet werden müsste, wird diese Zeitspanne nicht für die Migrationsdauer berücksichtigt, wenngleich der neue Operator erst dann sein erstes reguläres Ergebnis produziert. Die Ergebnisse der Experimente lassen sich so aber besser und vor allem fair miteinander vergleichen. Bei Fenstervergrößerungen, die mit zusätzlichen neuen Werten vervollständigt werden, wird weiterhin der letzte reguläre Wert zur Zeitnahme verwendet. Bei den dokumentierten Ergebnissen betrifft dies immer das letzte Experiment.

Mit Hilfe des heuristischen Verfahrens werden die Transferparameter anhand einer Berechnung von N_o ermittelt. N_o wiederum basiert auf der Abschätzung des mittleren Ereignisabstands des Eingangsdatenstroms. Für periodische Eingangsdatenströme lassen sich damit die Transferparameter exakt ermitteln. Bei aperiodischen Eingangsdatenströmen sind hingegen nur Näherungen für N_o und alle weiteren Parameter bestimmbar, was zu ungünstigen Effekten während der Migration führt. Die Konsequenzen werden an entsprechender Stelle erläutert.

Aufgrund der Abhängigkeit von der Abschätzung des mittleren Ereignisabstands des Eingangsdatenstroms wurden als Voraussetzung für die Evaluierung zwei aperiodische Datenstrommodelle in Hinblick auf die Genauigkeit der Abschätzung untersucht. Die Datenstrommodelle können sowohl im Prototyp als auch in der Simulation zur Erzeugung von Eingangsdatenströmen verwendet werden.

Mit „Random“ kann ein Datenstrom generiert werden, dessen Ereignisabstände auf einer Normalverteilung basieren. Des Weiteren steht mit „Poisson“ ein Modell zur Verfügung, dessen Ereignisse anhand eines Poisson-Prozesses erzeugt werden. Sowohl Datenströme in Sensornetzen [Neu06] als auch Vorgänge im Produktionsumfeld, z. B. bei zustandsorientierter Instandhaltung [Noo09], werden oft durch Poisson-Prozesse modelliert. Die Untersuchung der aperiodischen Datenstrommodelle wird durch einen schwankungsbeschränkten Datenstrom ergänzt.

Die Abschätzung des mittleren Ereignisabstands[16] des Eingangsdatenstroms während

[16] Aus Gründen der Übersichtlichkeit wird der Begriff „mittlerer Ereignisabstand $\overline{T}$“ hier auch für schwankungsbeschränkte Datenströme verwendet, obwohl für diese die „theoretische Periodendauer T“ festgelegt wurde. Semantisch sind beide Begriffe für schwankungsbeschränkte Datenströme im

der Laufzeit kann durch fortlaufende Beobachtung des Eingangsdatenstroms bewerkstelligt werden. Dies erhöht jedoch die Systemlast, insbesondere wenn mehrere Datenströme zu überwachen sind. Mit deutlich geringerem Aufwand kann der mittlere Ereignisabstand aus den inneren Zuständen der Operatoren gewonnen werden[17], indem die darin enthaltenen Datentupel und deren Zeitstempel für die Berechnung herangezogen werden, d. h. die Zeitstempel des ersten und des letzten Werts des Fensters sowie die Anzahl der Werte im Fenster.

Per Simulation wurde $\overline{T}$ für verschiedene Fenstergrößen für je 10000 Experimente bestimmt. Der erwartete mittlere Ereignisabstand, der als Parameter für jedes Modell angegeben wurde, ist $\overline{T} = 100\,\text{ms}$. Die maximalen Abweichungen für die untersuchten Fenstergrößen sind in Abbildung 7.24 für die drei Datenstrommodelle gegenübergestellt.

Für alle Datenstrommodelle verringern sich die Abweichungen mit wachsender Fenstergröße. Dies verdeutlichen auch die fallenden Verläufe in Abbildung 7.25, die jeweils den relativen Maximalwert des Abstands zu $\overline{T}$ darstellen. Die beste Abschätzung kann erwartungsgemäß für die schwankungsbeschränkten Datenströme (SBP) erreicht werden. Die aperiodischen Datenströme liegen etwa zwei bis drei Größenordnungen davon entfernt. Besonders für kleine Fenster ist die Spannweite groß.

Das nachfolgende Migrationsbeispiel verdeutlicht die Konsequenzen für die Abschätzung zur Laufzeit. Wird ein Fenster mit $w = 2500$ über einen Zeitraum von etwa 20 Sekunden migriert, dann werden während dieser Zeit ca. 200 neue Werte empfangen, bei einem Poisson-Eingangsdatenstrom mit $\overline{T} = 100\,\text{ms}$. Das Fenster mit 2500 wird für die Schätzung von $\overline{T}$ genutzt. Der tatsächliche mittlere Ereignisabstand während der Migration kann nach Migrationsende anhand der neuen Werte exakt bestimmt werden. In Abbildung 7.26 sind beide Größen für eine Serie von 5000 Experimenten abgebildet. Der geschätzte mittlere Ereignisabstand erstreckt sich dabei von 93 ms bis 109 ms, der tatsächliche mittlere Ereignisabstand jedoch von 77 ms bis 134 ms. Diese Werte entsprechen auch etwa den Ergebnissen der vorherigen Untersuchung für die Fenstergrößen $w = 2500$ bzw. $w = 200$. Die Ergebnismenge ist zufällig verteilt mit einer Häufung jeweils um $\overline{T}$.

Einzelfenster – Periodischer Eingangsdatenstrom

Um die Ergebnisse des heuristischen Verfahrens mit den Ergebnissen des numerischen Verfahrens direkt vergleichen zu können, wurden die Experimente aus Abschnitt 7.5.2 unter Verwendung der heuristischen Abschätzung wiederholt. Die Migration betrifft also ein einzelnes Fenster, wie in den Beispielanfragen RMS oder CF. Die Kollisionsbehandlung entspricht Szenario (a), d. h. die Migration verläuft verlustfrei. Der Eingangsdatenstrom ist periodisch. Die konstanten Experimentbedingungen sind wie zuvor:

- $T = 100\,\text{ms}$
- $N_i = 1$

hier beschriebenen Zusammenhang jedoch gleich.

[17] Operatoren müssen dafür eine geeignete Schnittstelle anbieten.

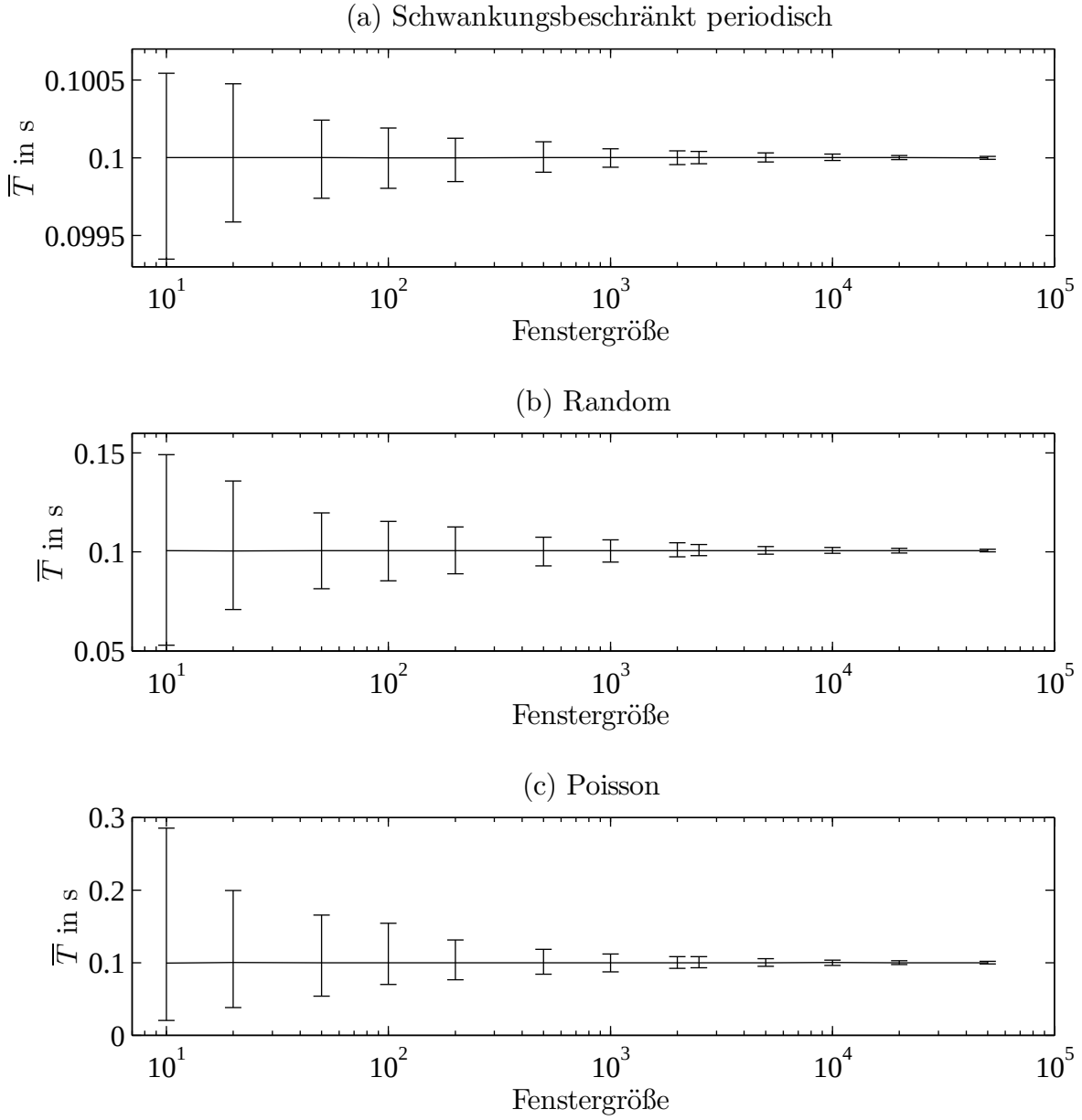


Abbildung 7.24: Untersuchung zur Abschätzung des mittleren Ereignisabstands für aperiodische Datenströme

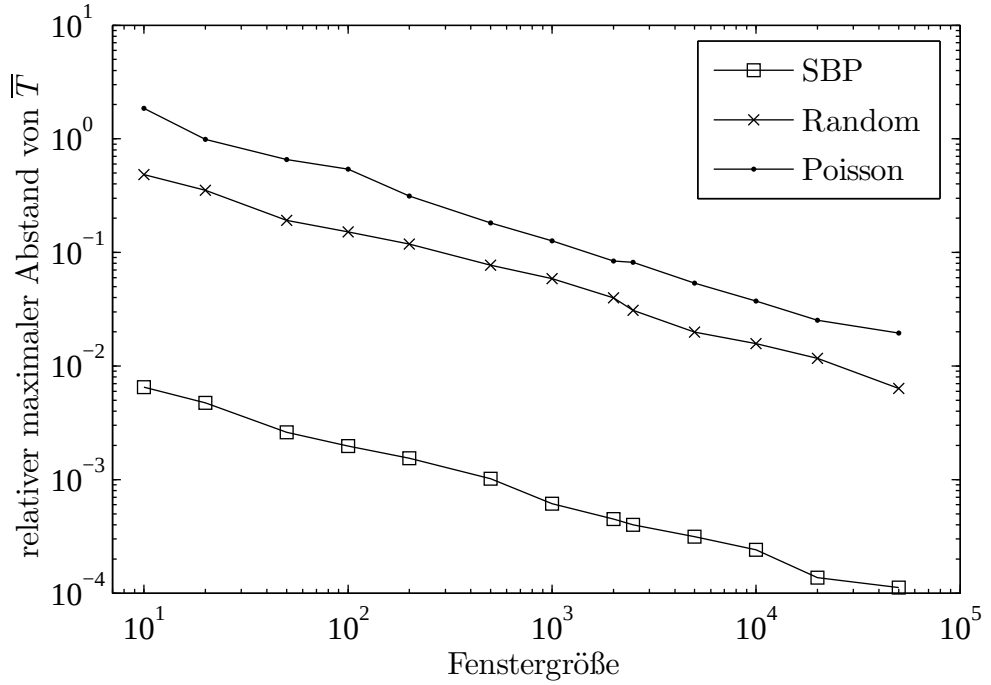


Abbildung 7.25: Untersuchung zur Abschätzung des mittleren Ereignisabstands für aperiodische Datenströme – relativer maximaler Abstand (normiert auf $\overline{T}$)

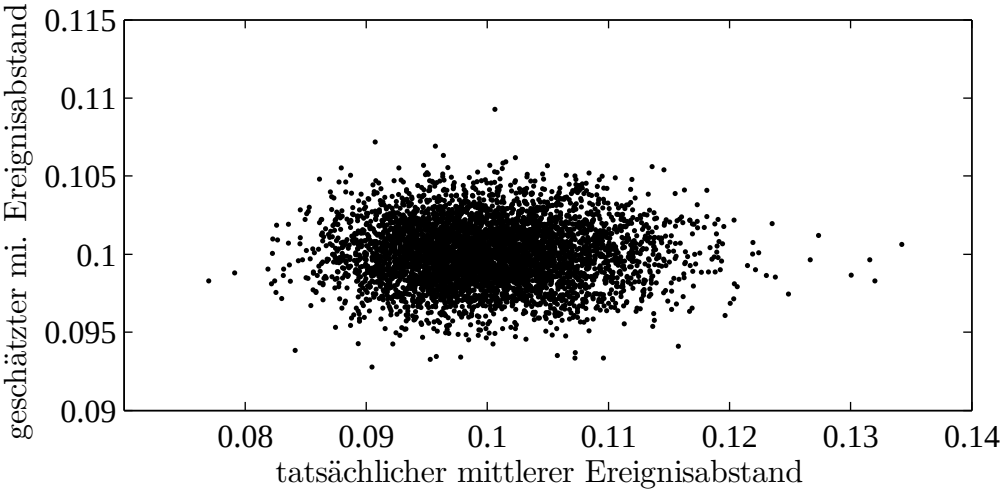


Abbildung 7.26: Geschätzter und tatsächlicher mittlerer Ereignisabstand im Experiment (Poisson)

- $T_p = 2\,\text{ms}$
- $T_t = 10\,\text{ms}$
- $w_o = 2500$

Variiert wurde die Größe des neuen Fensters:

- $w_n = 2500$ (gleichgroße Fenster)
- $w_n = 2000$ (Fensterverkleinerung)
- $w_n = 2600$ (Fenstervergrößerung)
- $w_n = 3000$ (Fenstervergrößerung)

Gleichgroße Fenster Tabelle 7.5 enthält die Ergebnisse für eine Migration zwischen gleichgroßen Fenstern ($w_o = w_n = 2500$). Die Zustandstransferdauer T_{ZT} wird bei diesem Experiment wie auch bei der Fensterverkleinerung nicht aufgelistet, da sie der Migrationsdauer T_M entspricht. Im Vergleich zu den Resultaten des numerischen Verfahrens (Tabelle 7.1) schließen die neuen Transferstrategien die Migration schneller ab, für die verwendete Konfiguration mehr als 4 s. Die Ursache dafür ist die Vernachlässigung der Reserve, weshalb mehr Zustandswerte während einer Pause übertragen werden können. Dies ist auch der Grund, weshalb die Migration mit GHM schneller durchgeführt werden kann. Die Migrationsdauer liegt jedoch weiterhin über der Migrationsdauer der Gruppe $\mathcal{G}$. Die erhöhte Transferquote während einer Pause führt auch dazu, dass insgesamt 47 zusätzliche Zustandswerte übertragen werden können. Für die verbleibenden Strategien GMS und GPT ändert sich nichts. Ihre Ergebnisse sind identisch zum numerischen Verfahren. Dies gilt auch für alle weiteren Ergebnisse von GMS und GPT in dieser Versuchsreihe.

Migrationsstrategie	$\mathcal{G}$	GMS	GPT	GHM
übertragene Werte (N_o)	2268	2500	0	2500
Migrationsdauer (T_M in s)	23,14	28,06	249,90	30,61
Ergebnisverzögerung (T_{OS} in s)	0,00	27,50	0,00	0,00

Tabelle 7.5: Vergleich verschiedener Migrationsstrategien bei einer Migration zwischen gleichgroßen Fenstern ($w_o = w_n = 2500$)

Fensterverkleinerung Auch bei einer Fensterverkleinerung (Tabelle 7.6) kann die Migration knapp 4 s eher beendet werden als bei der Anwendung des numerischen Verfahrens (vgl. Tabelle 7.2). Die Anzahl N_o erhöht sich für die gewählte Konfiguration um 38 Werte. Für GHM kann indes keine Verbesserung im Vergleich zum gleichgroßen Fenster beobachtet werden. Grund dafür ist die gleichbleibende Transfermenge.

Migrationsstrategie	$\mathcal{G}$	GMS	GPT	GHM
übertragene Werte (N_o)	1814	2500	0	2500
Migrationsdauer (T_M in s)	18,51	28,06	199,90	30,61
Ergebnisverzögerung (T_{OS} in s)	0,00	27,50	0,00	0,00

Tabelle 7.6: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fensterverkleinerung ($w_o = 2500$ und $w_n = 2000$)

Fenstervergrößerung Wie beim numerischen Verfahren lösen sich LF und RSD für Fenstervergrößerungen aus der gemeinsamen Gruppe der neuen Transferstrategien. Die verbleibenden Strategien bilden die Gruppe $\mathcal{G}_{OF}$. Die Ergebnisse von RSD befinden sich zwischen denen von $\mathcal{G}_{OF}$ und LF.

Die günstigsten Ergebnisse produzieren die Strategien der Gruppe $\mathcal{G}_{OF}$. Die Migrationsdauer ist die kleinste im Vergleich zu den anderen Strategien. Für LF sind 100 zusätzliche Werte notwendig, um das neue Fenster zu vervollständigen. Die Migration dauert dadurch ca. 10 s länger als der Zustandstransfer und übersteigt die Resultate der anderen Strategien, sieht man von GPT ab. Mit 2268 Werten ist für LF bei dieser Konfiguration die maximale Anzahl von übertragbaren Werten erreicht. Verglichen mit den Ergebnissen des numerischen Verfahrens (Tabelle 7.3) sind LF und $\mathcal{G}_{OF}$ jeweils schneller.

Migrationsstrategie	$\mathcal{G}_{OF}$	LF	GMS	GPT	GHM
übertragene Werte (N_o)	2359	2268	2500	0	2500
Zustandstransferdauer (T_{ZT} in s)	24,07	23,14	25,00	-	30,00
Migrationsdauer (T_M in s)	24,07	33,10	28,06	259,90	30,61
Ergebnisverzögerung (T_{OS} in s)	0,00	0,00	27,50	0,00	0,00
zusätzliche neue Werte ($N_{n,add}$)	0	100	0	-	0

Tabelle 7.7: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fenstervergrößerung ($w_o = 2500$ und $w_n = 2600$)

Die Fenstervergrößerung auf $w_n = 3000$ zeigt eine Konfiguration, bei der alle Strategien außer GPT auf zusätzliche neue Werte angewiesen sind (Tabelle 7.8). Die Migrationsdauer ist für $\mathcal{G}_{OF}$, GMS und GHM identisch, auch im Vergleich zum numerischen Verfahren (Tabelle 7.4) vernachlässigt man die Zeit von einer Periodendauer, die durch den sofortigen Beginn des Zustandstransfers gewonnen wird. Da der neue Operator sein erstes gültiges Ergebnis nach Abschluss der Migration erzeugt und dafür auf den nächsten regulären Eingangswert warten muss, ist die Migrationsdauer ungefähr gleich. Die Migrationsdauer für LF ist größer, da nicht alle Werte des alten Fensters übertragen werden können, insgesamt aber kleiner als mit dem numerischen Verfahren.

Für Fenstervergrößerungen mit der vorgegebenen Konfiguration sind für die Zustandstransferstrategien der Gruppe $\mathcal{G}_{OF}$ ab einer Fenstergröße von $w_n = 2757$ zusätzliche neue Werte notwendig, um das neue Fenster zu vervollständigen.

Migrationsstrategie	$\mathcal{G}_{OF}$	LF	GMS	GPT	GHM
übertragene Werte (N_o)	2500	2268	2500	0	2500
Zustandstransferdauer (T_{ZT} in s)	25,51	23,14	25,00	-	30,00
Migrationsdauer (T_M in s)	49,90	73,10	49,90	299,90	49,90
Ergebnisverzögerung (T_{OS} in s)	0,00	0,00	27,50	0,00	0,00
zusätzliche neue Werte ($N_{n,add}$)	244	500	220	-	193

Tabelle 7.8: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fenstervergrößerung ($w_o = 2500$ und $w_n = 3000$)

Zusammenfassung Mit dem heuristischen Verfahren können in periodischen Datenströmen kürzere Migrationszeiten als mit dem numerischen Verfahren erreicht werden, da der Zustandstransfer während der Pausen nicht vorzeitig unterbrochen wird sondern bis zum Eintreffen eines regulären Eingangswertes aufrecht erhalten wird. Je nach Kollisionsbehandlung entstehen so keine oder weniger Verluste von Transferzeit, was die Migration beschleunigt. Sofern Kollisionen im System toleriert und behandelt werden können, ist das heuristische Verfahren zu bevorzugen. Die Ergebnisse der existierenden Verfahren unterscheiden sich lediglich bei GHM, da diese Strategie ebenfalls die Zustandswerte während den Pausenzeiten transferiert. Die Ergebnisse von GMS und GPT sind unabhängig von den Verfahren zur Parameterbestimmung und deshalb für beide Experimentreihen identisch.

Einzelfenster – Aperiodischer Eingangsdatenstrom

Unter Verwendung der Beispielanfragen RMS und CF wurde das Verhalten der Migrationsstrategien auch für aperiodische Eingangsdatenströme analysiert. Für die nachfolgende Beschreibung der Evaluierungsergebnisse wurden Experimente ausgewählt, deren Eingangsdatenströme mit dem Poisson-Modell generiert wurden, da diese in der eingangs beschriebenen Untersuchung die größten Schwankungen zeigten und somit die geringste Genauigkeit bei der Abschätzung zur Folge haben, d. h. den ungünstigsten Fall repräsentieren. Die Ergebnisse anderer Konfigurationen und bei Verwendung anderer Datenstrommodelle sind qualitativ ähnlich. Die Parameter zur Konfiguration der Experimente wurden größtenteils beibehalten und wenn nötig auf den aperiodischen Datenstrom adaptiert. Das Kollisionsverhalten entspricht wieder Szenario (a). Gleichbleibende Experimentbedingungen sind:

- $\overline{T} = 100\,\text{ms}$ (Parameter zu Konfiguration des Eingangsdatenstroms)

- $N_i = 1$
- $T_p = 2\,\text{ms}$
- $T_t = 10\,\text{ms}$
- $w_o = 2500$

Variiert wurde die Größe des neuen Fensters:

- $w_n = 2500$ (gleichgroße Fenster)
- $w_n = 2000$ (Fensterverkleinerung)
- $w_n = 2600$ (Fenstervergrößerung)
- $w_n = 3000$ (Fenstervergrößerung)

Wie in der Einleitung zu diesem Abschnitt beschrieben, wird der mittlere Ereignisabstand aus Werten des Originalzustands ermittelt. Da daraus N_o und weitere Transferparameter berechnet werden, führt die Differenz zum tatsächlichen mittleren Ereignisabstand auch zum Über- oder Unterschätzen von N_o und somit zu Abweichungen bei den anderen Parametern. Der tatsächliche mittlere Ereignisabstand ist lediglich ein Vergleichswert für die Evaluierung und für reale Situationen irrelevant. Mit Hilfe dieses Wertes kann im Nachhinein die Konfiguration bestimmt werden, die unter den gegebenen Bedingungen zu einer optimalen Migration geführt hätte. Verglichen mit den tatsächlichen Migrationsergebnissen lassen sich so Rückschlüsse auf das Verhalten der einzelnen Transferstrategien ziehen.

Da durch die aperiodischen Eingangsdatenströme in jedem Experiment unterschiedliche Ergebnisse erzeugt werden, wurde jede Konfiguration 5000 Mal getestet. Deshalb wird für die Anzahl der übertragenen Werte (N_o), die Migrationsdauer (T_M) und die Ausgabeverzögerung (T_{OS}) jeweils das arithmetische Mittel aller Experimente angegeben. In den Ergebnistabellen wird dafür das Symbol *avg()* benutzt. Für die Migrationsdauer werden zusätzlich der Minimal- und Maximalwert angegeben, d. h. *min()* bzw. *max()*.

Gleichgroße Fenster Die Ergebnisse für die Migration zwischen gleichgroßen Fenstern in aperiodischen Datenströmen sind in Tabelle 7.9 aufgelistet. Im Unterschied zu den Experimenten in periodischen Datenströmen sind die Strategien der Gruppe $\mathcal{G}_{OF}$, RSD und LF in jedem Fall getrennt vertreten. Da LF ohne Abschätzung von N_o auskommt und Werte überträgt, solange sie verfügbar sind, werden mit LF die besten Migrationsergebnisse erzielt. Aus diesem Grund kann LF für alle Szenarien mit $w_o \geq w_n$ als Referenz genutzt werden. Die Werte für $\mathcal{G}_{OF}$ sind ähnlich den Werten von LF, einzig bei der maximalen Migrationsdauer ist ein deutlicher Unterschied zu beobachten. Dies ist

die Auswirkung des Unterschätzens von N_o. Dieser Effekt wird im Anschluss im Detail diskutiert.

Für die existierenden Migrationsstrategien sind die Ergebnisse vergleichbar mit den Werten des vorherigen Abschnitts. Die Spannweite für GMS und GHM ist relativ klein, da immer der komplette Zustand des Originaloperators übertragen wird. Bei GPT ist die Spannweite größer, was auf die stochastischen Eigenschaften des Poisson-Prozesses zurückzuführen ist. Gemessen an der Migrationsdauer ist mit LF das beste Migrationsergebnis für gleichgroße Fenster zu erzielen.

Migrationsstrategie	$\mathcal{G}_{OF}$	LF	GMS	GPT	GHM
avg(N_o)	2268,94	2268,91	2500	0	2500
min(T_M) in s	23,02	22,75	27,94	230,86	30,49
avg(T_M) in s	23,77	23,15	28,06	249,95	30,61
max(T_M) in s	29,69	23,57	28,19	268,64	30,74
avg(T_{OS}) in s	0,00	0,00	27,50	0,00	0,00

Tabelle 7.9: Vergleich verschiedener Migrationsstrategien bei einer Migration zwischen gleichgroßen Fenstern ($w_o = w_n = 2500$)

Durch die Unterschiede zwischen geschätztem und tatsächlichem mittleren Ereignisabstand entstehen für die Strategien der Gruppe $\mathcal{G}_{OF}$ Situationen, welche nachfolgend erläutert werden.

- *N_o wird unterschätzt:* Ist bei Ende des Zustandstransfers die Summe der transferierten und der neuen Werte kleiner als die Fenstergröße des neuen Fensters, wurde N_o unterschätzt. Die fehlenden Werte müssen dann durch neue Werte kompensiert werden[18], welche wesentlich langsamer eintreffen, als Zustandstransferwerte übertragen werden können. Deshalb kann das Unterschätzen von N_o zu erheblichen Verzögerungen der Migration führen. Im untersuchten Beispiel erhöht sich die Migrationsdauer um bis zu 30%.

- *N_o wird überschätzt:* Da der Transfer mit Strategien aus $\mathcal{G}_{OF}$ beim ältesten ausgewählten Wert beginnt, muss die gesamte Transfermenge übertragen werden, um ein lückenlos gefülltes Fenster zu erhalten. Auch wenn die Transferwerte zusammen mit den inzwischen empfangenen neuen Werten bereits die Fenstergröße ergeben, muss der Zustandstransfer bis zum neuesten Transferwert fortgesetzt werden. Ältere Transferwerte werden dann bereits verworfen. Die Zeit zum Vollenden des Transfers erhöht die Migrationsdauer geringfügig.

[18] Bei gleichgroßen Fenstern und bei Fensterverkleinerungen besteht auch die Möglichkeit, mit einem zweiten Transferzyklus alte Werte aus dem Originalfenster zu übertragen, da diese noch nicht verworfen wurden. Diese Variante wird hier jedoch nicht betrachtet.

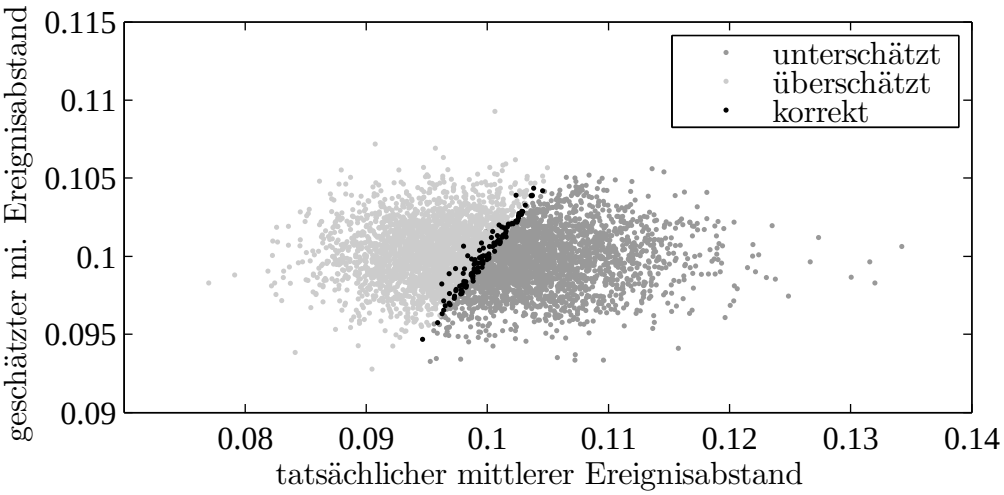


Abbildung 7.27: Ursache für das Überschätzen und Unterschätzen von N_o – geschätzter und tatsächlicher mittlerer Ereignisabstand für $\mathcal{G}_{OF}$ in aperiodischen Datenströmen

- *N_o wird korrekt geschätzt:* Sind geschätzter und tatsächlicher mittlerer Ereignisabstand nahezu gleich, ist die Wahrscheinlichkeit hoch, dass auch N_o und N_n den geschätzten Parametern entsprechen. Die Migration läuft dann ohne zusätzlichen Zeitaufwand für neue Werte oder Zustandstransfer ab.

In Abbildung 7.27 sind für den mittleren Ereignisabstand die geschätzten Werte über den tatsächlichen Werten für die Migrationsexperimente zwischen gleichgroßen Fenstern mit $w = 2500$ und der genannten Konfiguration dargestellt. Diese Darstellung entspricht Abbildung 7.26 mit dem Unterschied, dass hier die Ergebnisse entsprechend der verschiedenen Kategorien (unterschätzt, überschätzt, korrekt) markiert sind. Die Mengen der Ergebnisse für überschätzte und unterschätzte $\overline{T}$ sind etwa gleichgroß. Die Ergebnisse der korrekten Konfigurationen befinden sich da, wo geschätzter und tatsächlicher mittlerer Ereignisabstand nur geringfügig voneinander abweichen. Zum Unterschätzen (dunkelgraue Markierung) von N_o kommt es, wenn der tatsächliche mittlere Ereignisabstand größer ist, als zu Beginn der Migration anhand des inneren Zustands geschätzt wurde. Das bedeutet, während der Migration treffen weniger neue Werte ein als angenommen.

Stellt man die tatsächliche Migrationsdauer über dem Verhältnis von geschätztem und tatsächlichem mittleren Ereignisabstand dar (Abbildung 7.28), wird dieser Effekt und seine Konsequenzen deutlich. Während die Ergebnisse mit steigendem Überschätzen leicht ansteigen, steigt die Migrationsdauer der unterschätzen Fälle sehr stark mit wachsender Differenz zwischen geschätztem und tatsächlichem mittleren Ereignisabstand. Wenngleich ein Risiko einer extrem verlängerten Migration besteht, befinden sich mehr als 80% aller Ergebnisse dieser Menge im unteren Drittel bezogen auf die tatsächliche

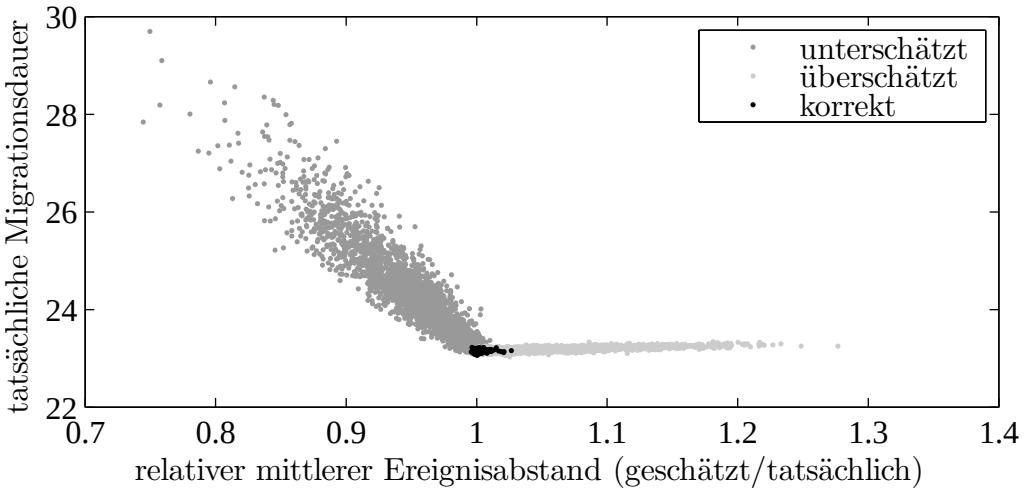


Abbildung 7.28: Auswirkung der Schätzung von N_o auf die Migrationsdauer bei Verwendung von $\mathcal{G}_{OF}$ in aperiodischen Datenströmen

Migrationsdauer. Die Ergebnisse der korrekten Konfigurationen häufen sich erwartungsgemäß bei einem ausgeglichenem Verhältnis (relativer mittlerer Ereignisabstand = 1).

Für alle Experimente dieses Abschnitts werden die Ergebnisse für die Strategien der Gruppe $\mathcal{G}_{OF}$ nach den unterschiedlichen Situationen aufgeschlüsselt. Dafür werden folgende Kategorien verwendet.

- Kategorie $\mathcal{C}_u$ – N_o wurde unterschätzt, die Migration musste mit zusätzlichen neuen Werten beendet werden
- Kategorie $\mathcal{C}_c$ – die abgeschätzte Menge von N_o war hinreichend, um die Migration mit dem Zustandstransfer zu beenden
- Kategorie $\mathcal{C}_o$ – N_o wurde überschätzt, der Zustandstransfer musste fortgeführt werden, um die Lücke im Fenster zu schließen

Die relative Häufigkeit jeder Kategorie und die verschiedenen Werte für die Migrationsdauer sind in Tabelle 7.10 aufgelistet. Wie auch Abbildung 7.27 wiedergibt, sind die Mengen von $\mathcal{C}_u$ und $\mathcal{C}_o$ etwa gleich groß, $\mathcal{C}_c$ nimmt nur einen Bruchteil der gesamten Menge ein. Die Minimalwerte von T_M sind für alle Kategorien nahezu identisch. Zudem zeigen die Werte für $\mathcal{C}_c$ und $\mathcal{C}_o$ eine geringe Spannweite zwischen Minimal- und Maximalwert. Bei $\mathcal{C}_u$ liegt der Maximalwert von T_M deutlich von den anderen Maximalwerten entfernt.

Fensterverkleinerung In Tabelle 7.11 sind die Ergebnisse der Fensterverkleinerung von $w_o = 2500$ auf $w_n = 2000$ dargestellt. Die Mittelwerte von N_o, T_M und T_{OS} für die Strategien LF, GMS, GPT und GHM sind annähernd gleich mit denen aus den Experimenten

Kategorie	$\mathcal{C}_u$	$\mathcal{C}_c$	$\mathcal{C}_o$
rel. Häufigkeit	0,483	0,023	0,495
$\min(T_M)$ in s	23,10	23,06	23,02
$\text{avg}(T_M)$ in s	24,40	23,15	23,19
$\max(T_M)$ in s	29,69	23,22	23,34

Tabelle 7.10: Aufschlüsselung für $\mathcal{G}_{OF}$, gleichgroße Fenster

mit dem periodischen Eingangsdatenstrom (vgl. Tabelle 7.6). Wie bereits zuvor erläutert, sind für GMS, GPT und GHM keine Verbesserungen bei Fensterverkleinerungen zu erreichen. Die Migrationsdauer der neuen Transferstrategien reduziert sich hingegen mit Verkleinerung der Fenstergröße w_n, wobei mit LF die besten Ergebnisse erzielt werden können. Die aufgeschlüsselten Ergebnisse für $\mathcal{G}_{OF}$ befinden sich in Tabelle 7.12. Die Anteile der einzelnen Kategorien sind nahezu unverändert im Vergleich zum vorherigen Experiment mit gleichgroßen Fenstern.

Migrationsstrategie	$\mathcal{G}_{OF}$	LF	GMS	GPT	GHM
$\text{avg}(N_o)$	1815,22	1814,87	2500	0	2500
$\min(T_M)$ in s	18,44	18,07	27,92	181,92	30,46
$\text{avg}(T_M)$ in s	19,08	18,52	28,06	200,00	30,61
$\max(T_M)$ in s	24,04	18,90	28,18	214,36	30,74
$\text{avg}(T_{OS})$ in s	0,00	0,00	27,50	0,00	0,00

Tabelle 7.11: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fensterverkleinerung ($w_o = 2500$ und $w_n = 2000$)

Kategorie	$\mathcal{C}_u$	$\mathcal{C}_c$	$\mathcal{C}_o$
rel. Häufigkeit	0,478	0,029	0,493
$\min(T_M)$ in s	18,48	18,46	18,44
$\text{avg}(T_M)$ in s	19,65	18,52	18,55
$\max(T_M)$ in s	24,04	18,58	18,69

Tabelle 7.12: Aufschlüsselung für $\mathcal{G}_{OF}$, Fensterverkleinerung

Fenstervergrößerung Die Ergebnisse für eine Fenstervergrößerung von $w_o = 2500$ auf $w_n = 2600$ sind in Tabelle 7.13 enthalten. Für die Strategie LF sind zusätzliche Werte

zum Vervollständigen des neuen Fensters nötig, weshalb die Migrationsdauer größer ist als bei $\mathcal{G}_{OF}$, GMS und GHM.

Betrachtet man die minimalen und durchschnittlichen Werte der Migrationsdauer, werden die besten Ergebnisse im dargestellten Fall durch Strategien der Gruppe $\mathcal{G}_{OF}$ erreicht. Die Strategien sind dabei rund 20% schneller als GMS oder GHM. Es besteht aber das Risiko, dass N_o unterschätzt wird, was eine vergleichsweise lange Migrationsdauer zur Folge hat. Diese kann unter Umständen auch die Werte von GMS oder GHM übersteigen. Die Wahrscheinlichkeit des Unterschätzens liegt laut Tabelle 7.14 bei 47,5%, die Wahrscheinlichkeit im konkreten Beispiel langsamer zu sein als GMS oder GHM[19] bei 1% bzw. 0,04%. Bei GMS sollte bei der Bewertung der Resultate jedoch berücksichtigt werden, dass für einen erheblichen Teil der Migration keine Ergebnisse ausgegeben werden.

Eine Möglichkeit zur Reduktion des Risikos N_o zu unterschätzen ist die Anpassung des Algorithmus zur Abschätzung des mittleren Ereignisabstands. Modifiziert man diesen so, dass der mittlere Ereignisabstand systematisch überschätzt wird, z. B. durch Multiplikation mit einem konstanten Faktor größer als 1, wirkt sich dies auch auf die Bestimmung von N_o aus. Zwar sind so mehr Zustandswerte zu übertragen, die Migrationsdauer erhöht sich durch den zusätzliche Zustandstransfer jedoch weniger schnell als durch das Warten auf zusätzliche Eingangswerte. Die Funktionsweise dieser einfachen Methode wurde durch Simulation bestätigt. Es wird empfohlen, die Optimierung dieser oder ähnlicher Strategien in zukünftigen Arbeiten aufzugreifen.

Migrationsstrategie	$\mathcal{G}_{OF}$	LF	GMS	GPT	GHM
avg(N_o)	2359,56	2268,90	2500	0	2500
min(T_M) in s	23,96	29,83	27,95	241,85	30,50
avg(T_M) in s	24,73	33,12	28,06	259,99	30,61
max(T_M) in s	31,42	36,95	28,18	279,97	30,74
avg(T_{OS}) in s	0,00	0,00	27,50	0,00	0,00

Tabelle 7.13: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fenstervergrößerung ($w_o = 2500$ und $w_n = 2600$)

Abschließend sind für die Migration von Einzelfenstern in aperiodischen Datenströmen die Resultate für eine Fenstervergrößerung von $w_o = 2500$ auf $w_n = 3000$ in Tabelle 7.15 dargestellt. Die Strategien aus $\mathcal{G}_{OF}$ sowie GMS und GHM erzielen bei der Migrationsdauer ungefähr die gleichen Ergebnisse. Das liegt wie in den vorherigen Szenarien an der Größe des neuen Fensters. Da die Werte des Originalfensters nicht ausreichen, um das neue Fenster zu füllen, muss auf neue Werte gewartet werden. Dies ist auch der Grund, weshalb es in diesem Fall bei $\mathcal{G}_{OF}$ nicht zum Überschätzen oder Unterschätzen kommt, der Inhalt des Originalfensters wird vollständig übertragen.

[19] verglichen zu den Mittelwerten

Kategorie	$\mathcal{C}_u$	$\mathcal{C}_c$	$\mathcal{C}_o$
rel. Häufigkeit	0,475	0,025	0,500
$\min(T_M)$ in s	24,03	23,96	23,96
$\text{avg}(T_M)$ in s	25,41	24,08	24,11
$\max(T_M)$ in s	31,42	24,18	24,28

Tabelle 7.14: Aufschlüsselung für $\mathcal{G}_{OF}$, Fenstervergrößerung $w_n = 2600$

Für alle Strategien entsprechen die Werte für die Anzahl der übertragenen Zustandswerte, für die durchschnittliche Migrationsdauer und für die durchschnittliche Ausgabeverzögerung ungefähr den Ergebnissen der Fenstervergrößerung von Einzelfenstern mit periodischen Eingangsströmen und heuristischer Abschätzung in Tabelle 7.8.

Migrationsstrategie	$\mathcal{G}_{OF}$	LF	GMS	GPT	GHM
$\text{avg}(N_o)$	2500	2268,65	2500	0	2500
$\min(T_M)$ in s	41,97	65,36	43,28	279,16	41,70
$\text{avg}(T_M)$ in s	49,93	73,17	49,96	299,98	49,97
$\max(T_M)$ in s	58,44	82,23	59,18	319,07	58,93
$\text{avg}(T_{OS})$ in s	0,00	0,00	27,50	0,00	0,00

Tabelle 7.15: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fenstervergrößerung ($w_o = 2500$ und $w_n = 3000$)

Zusammenfassung Wie bei der Verwendung des heuristischen Verfahrens mit periodische Datenströmen, können Migrationen mit den neuen Transferstrategien schneller beendet werden als mit den existierenden Strategien. Da der Eingangsdatenstrom nicht vorhersagbar ist, kommt es für die Strategien der Gruppe $\mathcal{G}_{OF}$ zum Überschätzen oder Unterschätzen von Transferparametern. Bei der gegebenen Konfiguration ist dies ungefähr bis zu einer Fenstergröße von $w_n = 2750$ der Fall. Besonders das Unterschätzen von N_o kann zu einer wesentlichen Verlängerung der Migration führen, eine Lösungsmöglichkeit wurde jedoch skizziert.

Mehrere Fenster

Die Evaluierung von Anfragen mit mehreren Fenstern wurde hauptsächlich anhand von Verbundbäumen durchgeführt, da diese auch von den existierenden Migrationsverfahren als primäre Evaluierungsmethode verwendet wurden (siehe u. a. [Zhu06, YKPS07]). Die existierenden Verfahren nutzen dabei immer zeitbasierte Fenster mit einer globalen Fenstergröße.

Die Funktionsweise der vorgestellten Migrationsstrategien wurde sowohl im Prototyp als auch simulativ mit verschiedenen Konfigurationen analysiert. Neben Migrationsszenarien mit zeitbasierten Fenstern wurde auch die Migration von anzahlbasierten Fenstern untersucht.

Die nachfolgend aufgelisteten Ergebnisse wurden mit Hilfe eines dreistufigen Verbundbaums wie in Abbildung 3.1 (S. 31) erzeugt. Die Migration betrifft die Zustände $\mathcal{Z}_A$, $\mathcal{Z}_B$, $\mathcal{Z}_C$, $\mathcal{Z}_D$, welche die Daten der Eingangsdatenströme enthalten. Die vier Eingangsdatenströme wurden auf Basis des Poisson-Modells generiert. Die Kollisionsbehandlungsmethode entspricht Szenario (a). Aufgrund der Vielzahl von Experimenten und der hohen Berechnungsdauer pro Experiment, die in der Simulation bereits schneller als in Echtzeit ablaufen, wurde die Anzahl auf jeweils 200 Wiederholungen reduziert. Die Ergebnisse sind jedoch hinreichend genau, um die Funktionsweise der Transferstrategien zu verstehen und zu validieren. Gleichbleibende Experimentbedingungen sind:

- $\overline{T} = 100\,\text{ms}$
- $N_i = 1$
- $T_p = 2\,\text{ms}$
- $T_t = 10\,\text{ms}$
- $w_o = 2500$
- $sel = 1/w_o$ bzw. $sel = 1/w_n$ (ein Ergebnis pro Eingangswert)

Variiert wurde die Größe des neuen Fensters:

- $w_n = 2500$ (gleichgroße Fenster)
- $w_n = 2000$ (Fensterverkleinerung)
- $w_n = 3000$ (Fenstervergrößerung)
- $w_n = 4000$ (Fenstervergrößerung)

Die Fenstergrößen sind anzahlbasiert und gelten für alle Transfer-relevanten Fenster als globale Fenstergrößen, d. h. die entsprechenden Fenster der Originalanfrage besitzen die Größe w_o, die Fenster der neuen Anfrage sind jeweils w_n groß. Durch die Verwendung von anzahlbasierten Fenstern mit der genannten Konfiguration lassen sich die Ergebnisse auch mit den Resultaten der Einzelfensterexperimente vergleichen, z. B. in Bezug auf die Anzahl der übertragenen Werte.

Der mittlere Ereignisabstand $\overline{T}$ bezieht sich auf die Konfiguration eines Eingangsdatenstroms. Das bedeutet, dass das Gesamtsystem durchschnittlich aller 25 ms einen neuen Eingangswert empfängt. Die Verarbeitungsdauer T_p ist die Zeit, die pro Operator zwischen Eingang eines Tupels und Ausgabe des entsprechenden Ergebnistupels vergeht.

Die Zeit wurde auf 2 ms festgelegt, um die Ergebnisse gegebenenfalls mit den Ergebnissen der Einzelfenster vergleichen zu können.

Ein Parameter des globalen Transferverhaltens ist die Reihenfolge der Zustände bei der Übertragung. Dafür wurden zwei verschiedene Methoden in Kombination mit der Strategie LF untersucht. Die einfachste Variante ist die Übertragung in einer fest definierten Reihenfolge, die während der Migration nicht verändert wird und somit statisch ist, z. B. A–B–C–D. Die Ergebnisse dieser Experimente sind in den nachfolgenden Tabellen unter LF_{stat} aufgeführt. Durch Beobachtung der neuen Fenster und der Anzahl der darin enthaltenen Werte während der Migration können Zustände beim Transfer priorisiert werden. So wurde eine Methode getestet, die immer den Zustand für den Zustandstransfer auswählt, der die wenigsten Werte im neuen Fenster besitzt. Die Reihenfolge der Zustandsübertragung ist dadurch dynamisch. Die entsprechenden Ergebnisse sind jeweils unter LF_{dyn} aufgelistet. Für die Strategien der Gruppe $\mathcal{G}_{OF}$ bringt eine dynamische Reihenfolge keine Vorteile, da die Transfermengen aller Fenster vollständig übertragen werden müssen. Aus diesem Grund wird für diese Transferstrategien eine statische Reihenfolge verwendet. Zustände, deren Transfer abgeschlossen wurde, werden in jedem Fall aus der Reihenfolge entfernt.

Wie bei der Übertragung von Einzelfenstern tritt bei den Strategien aus $\mathcal{G}_{OF}$ ein Über- oder Unterschätzen mit den bereits diskutierten Konsequenzen auf. Bei der Beschreibung des heuristischen Verfahrens (Abschnitt 6.2.2) wurde außerdem ein Effekt erläutert, der zu einem systematischen Überschätzen von N_n und folglich zu einem Unterschätzen von N_o führt. Zur Kompensation des Effekts wurde ein einfaches Verfahren entwickelt, welches nach der ersten Abschätzung der Migrationsparameter die potentiell zu viel geschätzte Transferzeit ermittelt. Diese Zeit wird von der geschätzten Transferzeit abgezogen und die endgültigen Transferparameter dann auf Basis der modifizierten Transferzeit berechnet.

In Abbildung 7.29 sind die Ergebnisse einer Untersuchung für die Migration mit oben genannten Parametern und für verschiedene Größen des neuen Fensters dargestellt. Die berechnete überschüssige Transferzeit wird zudem durch einen Faktor f im Bereich $[0, 2]$ gewichtet und von der Gesamttransferzeit abgezogen. Bei $f = 0$ findet keine Anpassung statt. Die so konfigurierten Anfragen wurden mit 200 Wiederholungen getestet und die gemessene Migrationsdauer gemittelt.

Die Ergebnisse zeigen, dass bis zu einer Fenstergröße von $w_n = 2750$ die schnellste Migration mit $f = 1$ erreicht wird. Für größere Fenster bewegt sich der Minimalwert von T_M stetig in Richtung $f = 0{,}25$, wobei die größte Abweichung bei $w_n = 3500$ auftritt. In diesem Fall liegt die minimale Migrationsdauer bei $f = 0{,}5$, für $f = 1$ ist T_M ca. 4% größer. Der Maximalwert von T_M liegt für alle Versuche bei $f = 0$. Ab einer Fenstergröße von $w_n = 3900$ ist kein Einfluss zu beobachten. Die Ursache ist, dass für diesen Fall bereits zusätzliche neue Tupel zur Vervollständigung der Fenster benötigt werden. Da die Migration mit $f = 1$ in den meisten Fällen die besten Ergebnisse liefert oder nur geringfügig von den besten Ergebnissen abweicht, werden nachfolgend die Ergebnisse für diese Konfiguration unter $\mathcal{G}_{OF,1}$ aufgelistet. Zum Vergleich werden auch die Ergebnisse

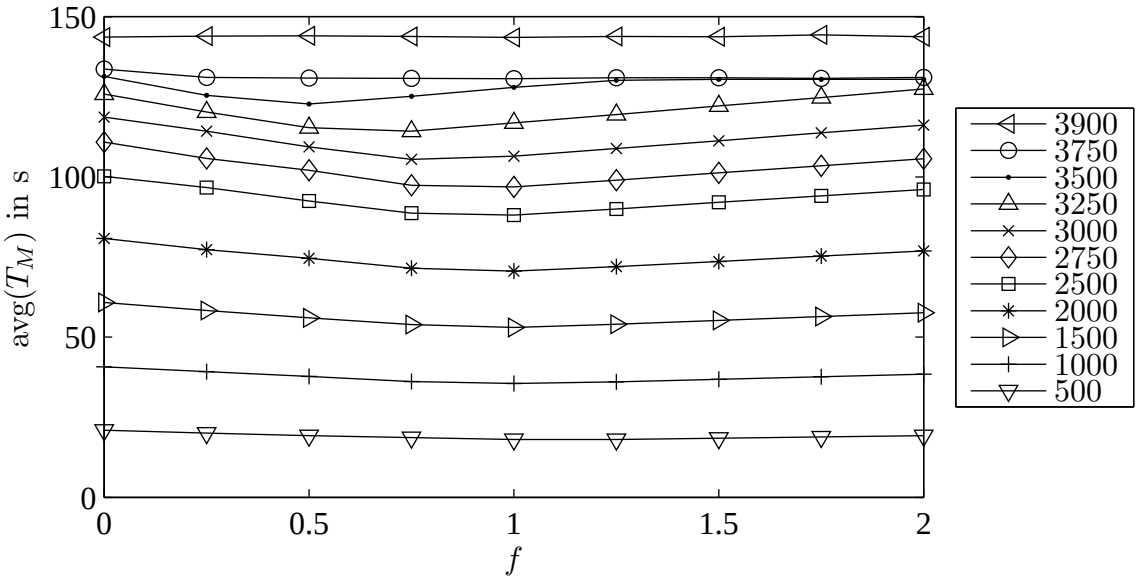


Abbildung 7.29: Einfluss des Korrekturfaktors f bei Migration mit Strategien der Gruppe $\mathcal{G}_{OF}$

ohne Korrektur angegeben, diese unter $\mathcal{G}_{OF,0}$.

Gleichgroße Fenster Die Ergebnisse für eine Migration zwischen Verbundbäumen mit gleichgroßen Fenstern sind in Tabelle 7.16 angegeben. Die Anzahl der übertragenen Werte wird in diesem Fall durch eine weitere Größe ergänzt. Das bedeutet, avg(N_o) steht für die durchschnittliche Anzahl übertragener Werte, gemittelt über alle Zustände und über alle Experimente. Die Summe aller Transferwerte, gemittelt über alle Experimente, ist unter avg($\mathcal{N}_o$) angegeben. Da mit der genannten Konfiguration jeweils vier Zustände transferiert werden, gilt avg($\mathcal{N}_o$) = $4 * \text{avg}(N_o)$. Um die Ergebnisse mit den vorherigen Experimenten vergleichen zu können, werden beide Werte angegeben.

Die besten Migrationsergebnisse für gleichgroße Fenster werden mit LF und dynamischer Zustandsreihenfolge (LF_{dyn}) erreicht. Die statische Variante ist nur minimal langsamer, deutlich weniger als 1%. Die Strategien aus $\mathcal{G}_{OF}$ sind durchschnittlich nur etwa 2,5 s langsamer als LF_{dyn} wenn der Korrekturfaktor $f = 1$ verwendet wird. Ohne diesen dauert die Migration deutlich länger. Zudem zeigt der Vergleich der Maximalwerte bei $\mathcal{G}_{OF,0}$ eine größere Abweichung zum Durchschnittswert, d. h. einen größeren negativen Effekt hervorgerufen durch das Unterschätzen von N_o. Dies wird auch anhand von avg($\mathcal{N}_o$) deutlich. Nimmt man den Wert von LF_{dyn} als Optimum, liegt $\mathcal{G}_{OF,0}$ 437 Werte darunter, was die Migrationsdauer beträchtlich verlängert. $\mathcal{G}_{OF,1}$ liegt hingegen 181 Werte darüber, die Auswirkungen auf die Migrationsdauer sind jedoch wesentlich geringer.

Etwa 40% mehr Zeit im Vergleich zu LF_{dyn} wird für die Migration mit GMS benötigt,

bei GHM sogar über 50%. Die Übertragung der vollständigen Originalzustände macht sich hier deutlich bemerkbar, 6567 Werte bei LF_{dyn} stehen 10000 Werten bei GMS oder GHM gegenüber.

Migrationsstrategie	$\mathcal{G}_{OF,0}$	$\mathcal{G}_{OF,1}$	LF_{stat}	LF_{dyn}	GMS	GPT	GHM
avg(N_o)	1532,5	1687,2	1646,7	1641,9	2500	0	2500
avg($\mathcal{N}_o$)	6129,8	6748,8	6586,7	6567,4	10000	0	10000
min(T_M) in s	94,10	87,38	84,91	84,66	119,16	244,81	130,00
avg(T_M) in s	100,14	88,10	85,91	85,68	119,57	255,19	130,44
max(T_M) in s	107,54	92,46	87,07	86,76	119,99	266,86	130,96
avg(T_{OS}) in s	0,00	0,00	0,00	0,00	110,00	0,00	0,00

Tabelle 7.16: Vergleich verschiedener Migrationsstrategien bei einer Migration zwischen gleichgroßen Fenstern ($w_o = w_n = 2500$)

Fensterverkleinerung Auch bei einer Fensterverkleinerung, hier am Beispiel von $w_o = 2500$ nach $w_n = 2000$ (Tabelle 7.17), wird die schnellste Migration mit LF_{dyn} realisiert. Die Differenzen zu LF_{stat}, $\mathcal{G}_{OF,0}$ und $\mathcal{G}_{OF,1}$ sind ähnlich groß wie im vorherigen Experiment. Der Abstand zu GMS und GHM wird hingegen größer, da diese Strategien etwa doppelt so viele Werte übertragen müssen wie LF_{dyn}. Die Migrationsdauer von GPT verringert sich aufgrund der kleineren neuen Fenster. Bei der gegebenen Konfiguration ist GPT für Fensterverkleinerungen unter $w_n = 1300$ sogar schneller als GHM und unter $w_n = 1190$ auch schneller als GMS. Die neuen Transferstrategien können von GPT allerdings nicht unterboten werden.

Migrationsstrategie	$\mathcal{G}_{OF,0}$	$\mathcal{G}_{OF,1}$	LF_{stat}	LF_{dyn}	GMS	GPT	GHM
avg(N_o)	1226,1	1349,8	1317,1	1314,2	2500	0	2500
avg($\mathcal{N}_o$)	4904,5	5399,2	5268,4	5256,9	10000	0	10000
min(T_M) in s	74,10	70,02	67,62	67,41	119,19	196,78	129,91
avg(T_M) in s	80,80	70,61	68,73	68,57	119,55	204,77	130,41
max(T_M) in s	88,22	73,98	70,04	69,48	119,91	216,72	130,84
avg(T_{OS}) in s	0,00	0,00	0,00	0,00	110,00	0,00	0,00

Tabelle 7.17: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fensterverkleinerung ($w_o = 2500$ und $w_n = 2000$)

Fenstervergrößerung Tabelle 7.18 enthält ein Beispiel einer Fenstervergrößerung mit $w_o = 2500$ und $w_n = 3000$. Das beste Migrationsergebnis wird mit $\mathcal{G}_{OF,1}$ erreicht. Vor

allem die Kompensation der zu viel geschätzten Transferzeit sorgt bei der tatsächlichen Migrationsdauer für einen deutlichen Abstand zu allen anderen Strategien. Die Migration mit LF benötigt dagegen merklich länger, wie dies auch bei den vorherigen Experimenten zu beobachten war. Die Ursache ist wie zuvor, dass für LF die maximale Anzahl von übertragbaren Werten bei $w_o = w_n$ erreicht ist und für Fenstervergrößerungen immer zusätzliche neue Werte zum Auffüllen benötigt werden, im Beispiel ca. 500 Werte pro Zustand.

Migrationsstrategie	$\mathcal{G}_{OF,0}$	$\mathcal{G}_{OF,1}$	LF_{stat}	LF_{dyn}	GMS	GPT	GHM
avg(N_o)	1854,2	2040,0	1647,1	1642,4	2500	0	2500
avg($\mathcal{N}_o$)	7416,8	8160,2	6588,3	6569,7	10000	0	10000
min(T_M) in s	111,70	105,39	134,15	133,76	119,17	295,91	129,96
avg(T_M) in s	118,71	106,46	137,73	137,89	119,57	305,13	130,44
max(T_M) in s	127,19	108,00	141,88	143,86	119,91	322,86	130,80
avg(T_{OS}) in s	0,00	0,00	0,00	0,00	110,00	0,00	0,00
avg($N_{n,add}$)			494,3	499,6		-	

Tabelle 7.18: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fenstervergrößerung ($w_o = 2500$ und $w_n = 3000$)

Das abschließende Beispiel mit $w_o = 2500$ und $w_n = 4000$ (Tabelle 7.19) ist aus dem Bereich, wo alle Strategien auf zusätzliche neue Werte angewiesen sind, um die neuen Fenster zu vervollständigen. Wie in allen vorherigen, vergleichbaren Experimenten erreichen die Strategien der Gruppe $\mathcal{G}_{OF}$ (egal ob mit oder ohne Korrektur) gemeinsam mit GMS und GHM die besten Ergebnisse in Bezug auf die Migrationsdauer. Die Experimente mit LF dauern für die verwendete Konfiguration ca. 86 s länger. Die beiden Varianten von LF unterscheiden sich dabei nur marginal. Die meiste Zeit wird für eine Migration mit GPT benötigt.

Zusammenfassung Die Evaluierung mit den Verbundbäumen hat gezeigt, dass die neuen Transferstrategien auch in Anfragen mit mehreren Fenstern bessere Migrationsergebnisse als die existierenden Strategien ermöglichen. Eine Übersicht über die Migrationsdauer unter Einbeziehung weiterer Experimente mit verschiedenen Fenstergrößen (w_n) ist in Abbildung 7.30 dargestellt. Einen Ausschnitt über den Intervall $w_n = 2200$ bis $w_n = 4000$ zeigt Abbildung 7.31.

Prinzipiell ähneln sich die Abbildungen mit den Ergebnissen aus der Untersuchung des numerischen Verfahrens (vgl. Abbildungen 7.21 und 7.22). Solange die neue Fenstergröße kleiner oder gleich der alten Fenstergröße ist, wird mit LF die schnellste Migration erreicht. Danach werden mit $\mathcal{G}_{OF,1}$ die besten Ergebnisse erzielt. Ist eine Kompensation bei der Berechnung der Transferparameter nicht möglich, kann LF etwas länger genutzt

Migrationsstrategie	$\mathcal{G}_{OF,0}$	$\mathcal{G}_{OF,1}$	LF_{stat}	LF_{dyn}	GMS	GPT	GHM
avg(N_o)	2500,0	2500,0	1646,2	1642,8	2500	0	2500
avg($\mathcal{N}_o$)	10000,0	10000,0	6584,7	6571,0	10000	0	10000
min(T_M) in s	147,74	147,57	233,05	232,57	145,71	397,64	147,98
avg(T_M) in s	153,55	154,16	239,61	239,68	153,88	406,95	153,75
max(T_M) in s	162,35	161,75	245,28	249,17	162,12	423,79	162,24
avg(T_{OS}) in s	0,00	0,00	0,00	0,00	110,00	0,00	0,00
avg($N_{n,add}$)	192,8	196,2	1494,6	1499,7	305,1	-	194,7

Tabelle 7.19: Vergleich verschiedener Migrationsstrategien bei einer Migration mit Fenstervergrößerung ($w_o = 2500$ und $w_n = 4000$)

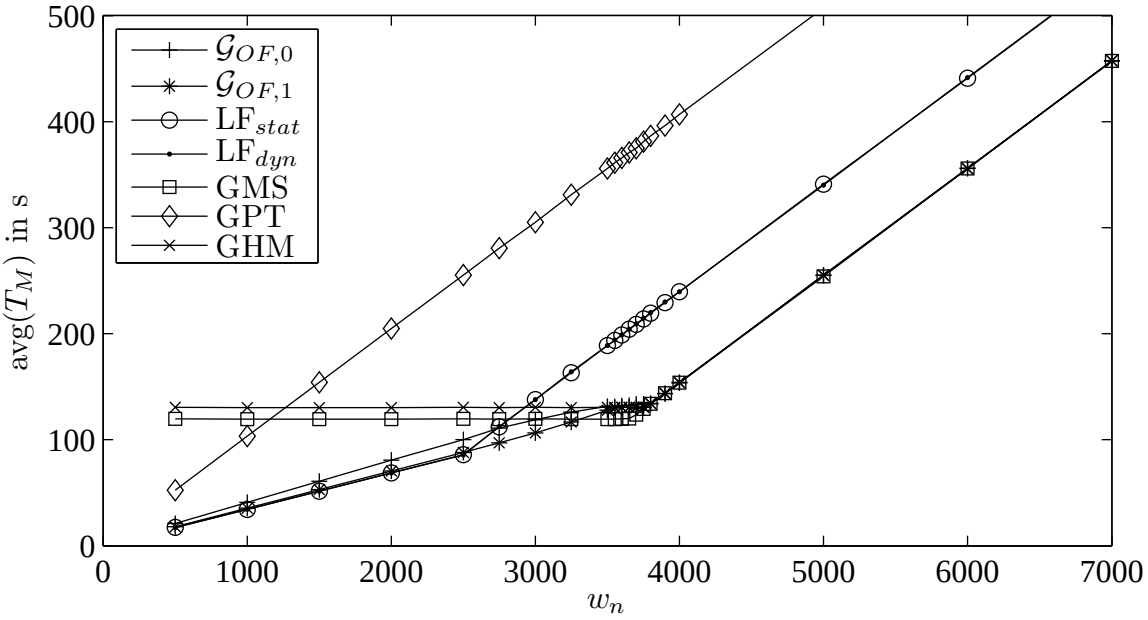


Abbildung 7.30: Migrationsdauer für verschiedene Migrationsstrategien

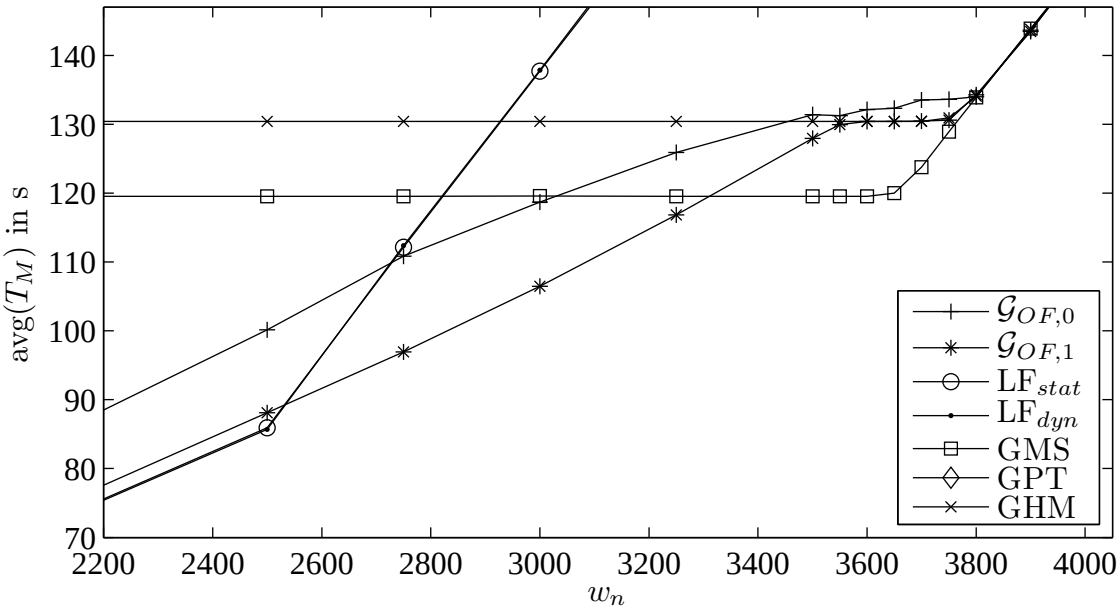


Abbildung 7.31: Migrationsdauer (Ausschnitt) für verschiedene Migrationsstrategien

werden, ehe $\mathcal{G}_{OF,0}$ die besseren Ergebnisse liefert. Im dargestellten Beispiel ist dies bei einer neuen Fenstergröße von ca. 2750 Werten der Fall.

Für eine neue Fenstergröße zwischen 3300 und 3800 wird die kürzeste Migrationsdauer mit GMS erreicht. Man darf jedoch nicht vergessen, dass dann für einen Großteil der Migration keine Ergebnisse ausgegeben werden. Ist dies, wie in den Anforderungen formuliert, nicht zulässig, sondern wird eine kontinuierliche Ergebnisausgabe verlangt, kann GMS nicht genutzt werden.

Gemessen an der Migrationsdauer werden im dargestellten Beispiel unter Einhaltung der Anforderungen die besten Ergebnisse durch die neuen Transferstrategien erreicht. Insbesondere bei Fensterverkleinerungen haben diese beträchtliche Vorteile gegenüber den existierenden Verfahren. Auch bei Fenstervergrößerungen werden bessere Resultate erzielt, wobei sich der Abstand fortlaufend verringert. Ab einer bestimmten Grenze, im Beispiel bei $w_n \approx 3900$, ist die Migrationsdauer für $\mathcal{G}_{OF}$, GMS und GHM gleich. Im Bereich vor dieser Grenze, im Beispiel zwischen 3000 und 3900, kann die Migrationsdauer für $\mathcal{G}_{OF}$ durch die Verbesserung der Korrekturmethode weiter verringert werden. Das Optimierungspotential liegt hier bei ca. 5%.

Verbundbäume repräsentieren natürlich nur einen kleinen Teil von Anfragen mit mehreren Fenstern, deren Menge aufgrund der Kombinations- und Konfigurationsmöglichkeiten potentiell unendlich ist. Deshalb sollten alle Migrationsstrategien weiter systematisch untersucht werden. So haben Untersuchungen im Prototyp gezeigt, dass Verbundbäume mit zeitbasierten Fenstern oder mit größeren Selektivitäten qualitativ vergleichbare Migrationsergebnisse erzeugen. Ebenso wurden Konfigurationen mit heterogenen Fens-

tern stichprobenartig untersucht. Um allgemeingültige Aussagen zu treffen, ist jedoch eine systematische Analyse notwendig. Vor allem das heuristische Verfahren bietet eine geeignete Grundlage für zukünftige Forschungsarbeiten.

7.6 Zusammenfassung

Zur Untersuchung der Eigenschaften aller Transferstrategien wurden sowohl Simulationen als auch ein funktionierender Prototyp entwickelt. Der Prototyp basiert auf dem Java-Modul-System OSGi und ermöglicht das Nachladen und den Austausch von Programmcode ohne Unterbrechung der laufenden Verarbeitung. Damit können Datenstromoperatoren mit beliebiger Verarbeitungslogik aktualisiert werden. Zu Beginn des Kapitels sind Details zur Implementierung des Prototyps beschrieben. Zudem wird auf die Laufzeiteigenschaften und deren Auswirkungen auf die Evaluierung eingegangen. Des Weiteren werden die Randbedingungen zur Durchführung der Experimente erläutert. Dazu zählen Betrachtungen zur Korrektheit der Transfermethoden sowie allgemeine Festlegungen zu Messgrößen und Experimentbedingungen.

Die Untersuchung der Transfermethoden konzentriert sich zuerst auf das Laufzeitverhalten der neuen Strategien. Dafür wurde anhand von Beispielanfragen und -konfigurationen das Transferverhalten beobachtet und diskutiert. Ein besonderes Augenmerk liegt dabei auf der Übertragungsreihenfolge der Zustandswerte und deren Auswirkungen auf die Migrationsdauer. Eine umfassende Analyse der Strategie SemS schließt diesen Abschnitt ab. Für alle Strategien konnte eine korrekte Funktionsweise gezeigt werden.

Abschließend wurde das Laufzeitverhalten der entwickelten Zustandstransferstrategien mit existierenden Migrationsstrategien verglichen, insbesondere in Bezug auf die Migrationsdauer und die Ergebnisverzögerung. Dafür wurde zunächst das numerische Verfahren in periodischen und schwankungsbeschränkten Datenströmen untersucht. Der anschließende Vergleich unter Verwendung des heuristischen Verfahrens für alle Datenstromtypen und für Einzelfenster sowie für Anfragen mit mehreren Fenstern schließt die Performance-Analyse ab. Hierbei entstehen besonders bei Verwendung des heuristischen Verfahrens in aperiodischen Datenströmen interessante Situationen, z. B. durch Über- oder Unterschätzen von Parametern, deren Konsequenzen auch zukünftig systematisch untersucht werden sollten, um Migrationsvorgänge in Datenstromsystemen weiter zu optimieren.

Die Ergebnisse der verschiedenen Untersuchungen zeigen, dass die neu entwickelten Transferstrategien schnellere Migrationen erlauben, zum Teil mit erheblichen Zeitvorteilen gegenüber den existierenden Verfahren. Die Strategien genügen dabei den gestellten Anforderungen, d. h. sie ermöglichen eine schnelle Migration bei gleichzeitiger kontinuierlicher Ergebnisausgabe. Der vorgestellte Ansatz und die entwickelten Verfahren zur Bestimmung der Migrationsparameter sollten als Basis für weitere Untersuchungen zur Migrationsoptimierung genutzt werden.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Die Überwachung von Prozessen und Anlagen, beispielsweise in Produktionsumgebungen, wird durch eine stetig steigende Anzahl von Sensoren und Messpunkten, durch höhere Abtastraten und durch vielfältige Methoden zur Aggregation von Daten realisiert – vermehrt auch in Echtzeit. Um die damit verbundene wachsende Menge von Daten zu beherrschen, werden die Daten oft direkt verarbeitet, ohne die Rohdaten zuvor in Datenbanken abzulegen. Dies wird z. B. durch Datenstromsysteme realisiert, welchen die Daten in Form von Datenströmen mit meistens kontinuierlich laufenden Datenstromanfragen im Arbeitsspeicher eines Rechensystems verarbeiten. Verarbeitete Daten, die nicht weiter relevant sind, werden aus dem Arbeitsspeicher verworfen und sind fortan nicht mehr zugänglich. Datenstromanfragen setzen sich aus Datenstromoperatoren zusammen, welche gegebenenfalls die empfangenen Daten in einem inneren Zustand speichern. Zur Festlegung, welche und wie viele Daten im Zustand gespeichert werden, nutzt man sogenannte Fenster. Für eine effiziente Verarbeitung der Anfragen werden die Operatoren typischerweise verteilt ausgeführt.

Da die kontinuierlichen Datenstromanfragen potentiell unendlich lange ausgeführt werden, sich während ihrer Laufzeit aber Anforderungen ändern können, ist ein Verfahren notwendig, um die Anfragen während der Laufzeit an die geänderten Bedingungen anzupassen. Typische Anwendungsfälle sind Anfrageoptimierung, Lastverteilung oder Parallelisierung zur Steigerung der Effizienz aber auch Fehlersicherheit und Neuinitialisierung von Operatoren und Anfragen. Dabei sind insbesondere zustandsbehaftete Operatoren eine große Herausforderung für die Durchführung während der Laufzeit, da ihre inneren Zustände im Arbeitsspeicher flüchtig sind und somit beim Beenden der Operatoren verloren gehen. Beim Neustart von zustandsbehafteten Operatoren ist dann mit einer Anlaufzeit zu rechnen, in der die inneren Zustände wieder aufgefüllt werden, was je nach Zustandsgröße, Fenstertyp und Datenstromeigenschaften lange Zeit in Anspruch nehmen kann. Bei der Anpassung von Anfragen sollten deshalb die Operatorzustände möglichst erhalten werden.

Für diesen Zweck wurden in vorherigen Arbeiten Migrationsverfahren entwickelt und beschrieben, welche durch Funktionen wie Parallelausführung oder Zustandstransfer

für einen Übergang von einer alten zu einer neuen Anfrage sorgen. Die Analyse der existierenden Migrationsverfahren zeigte jedoch, dass diese Einschränkungen aufweisen. So wird beispielsweise die Ergebnisausgabe während der Migration unterbrochen oder Ergebnisse in einer vertauschten Reihenfolge ausgegeben. In anderen Fällen dauert die Migration so lange wie die Anlaufzeit oder Migrationsverfahren sind nicht verteilt anwendbar. Deshalb wurde in dieser Arbeit ein Migrationsverfahren entwickelt, welches einen Zustandstransfer bei gleichzeitiger Ergebnisausgabe erlaubt. Die Übertragung kann durch verschiedene Zustandstransfermethoden realisiert werden und beschränkt sich auf notwendige Zustandswerte, wodurch in vielen Fällen eine verkürzte Migration erreicht wird. Das neue Migrationsverfahren ist sowohl zentral als auch verteilt anwendbar. Im Unterschied zu den existierenden Arbeiten, welche sich in erster Linie mit der Anfrageoptimierung befassen, stand in dieser Arbeit die Optimierung der Migration im Vordergrund.

Für die Konfiguration und Durchführung einer Migration wurde eine allgemeine Adaptionsmethodik für laufende Datenstromanfragen entwickelt. Alle weiteren beschriebenen Konzepte integrieren sich in dieses Verfahren. Da die fehlerfreie Ausführung der regulären Datenstromanfragen von höchster Wichtigkeit ist, wird das System logisch in Primär- und Testsystem unterteilt. Mögliche Änderungen können so im Testsystem erprobt werden, bevor sie mittels Migration in das Primärsystem integriert werden. Die Adaptionsmethodik umfasst drei Schritte zur Erzeugung und Anpassung von Datenstromanfragen sowie zur Durchführung der Migration. Dabei werden im Vergleich zu existierenden Verfahren neben Kompositionsänderungen erstmals auch Parameter- und Algorithmusänderungen von Operatoren unterstützt. Eine besondere Rolle während der Migration nimmt die Konfiguration des Zustandstransfers ein, weshalb diese in separaten Kapiteln beschrieben wurde.

Zunächst wurde für die Anwendung des Zustandstransfers in periodischen und schwankungsbeschränkten Datenströmen ein numerisches Verfahren zur Ermittlung der Migrationsparameter erläutert. Betrachtet wurden dabei sowohl Einzelzustände als auch Anfragen mit mehreren Zuständen. Für den zweiten Fall wurden Kriterien beschrieben, um relevante Zustände für den Transfer auszuwählen, da nicht immer alle Zustände einer Anfrage übertragen werden müssen. Dafür wurden die Eigenschaften von Zuständen, Operatoren, Anfragen und des Systems analysiert. Auch für die ausgewählten Zustände selbst gilt, dass nicht immer alle Werte eines Zustands übertragen werden müssen. Da die Migration in einem verteilten System eine gewisse Zeit in Anspruch nimmt und während dieser Dauer auch neue Werte im System eintreffen, ist es meistens ausreichend, nur eine Teilmenge des alten Zustands zu übertragen – ein neuer Ansatz gegenüber den existierenden Migrationsverfahren. Zur Bestimmung dieser Transfermenge wurde ein numerisches Verfahren für verschiedene Systemvarianten und mit einigen Erweiterungsmöglichkeiten beschrieben. Dieses berücksichtigt im Vergleich zu existierenden Migrationsverfahren auch die Zeit, die für die Übertragung von Zustandswerten in verteilten Umgebungen notwendig ist.

Für die ausgewählten Zustandswerte wurden Zustandstransferstrategien entwickelt,

mit denen z. B. unterschiedliche Übertragungsreihenfolgen realisiert werden können. Eine besondere Rolle nimmt dabei eine semantische Strategie (SemS) ein, mit der Migrationen ausgeführt werden können, bei denen sehr schnell zur neuen Anfrage umgeschaltet werden kann. Ob diese Strategie eingesetzt werden kann, hängt im Wesentlichen vom Operatortyp ab. Es konnte gezeigt werden, dass sich einige Aggregationsoperatoren sehr gut für eine Migration mit SemS eignen. Die endgültige Konfiguration jeder Migration wird als lokales und globales Transferverhalten gespeichert und im System entsprechend ausgeführt. Die Beschreibung der dafür notwendigen funktionalen Erweiterungen von Operatoren und des Datenstromsystems vervollständigt das Kapitel über den Zustandstransfer.

Aufbauend auf dem numerischen Verfahren wurde ein heuristisches Verfahren für die Verwendung mit aperiodischen Datenströmen entwickelt. Die Heuristik ist jedoch auch für periodische und schwankungsbeschränkte Datenströme anwendbar. Das Verfahren ermittelt die Migrationsparameter basierend auf der Abschätzung von Datenstrom- und Zustandseigenschaften und wurde für Einzelfenster und für Anfragen mit mehreren Fenstern beschrieben. In aperiodischen Datenströmen kann die Verwendung einiger Zustandstransferstrategien zu unerwünschten Effekten führen, welche in diesem Kapitel kurz diskutiert wurden. Eine umfassende Analyse dieser Effekte wurde in der Evaluierung durchgeführt und Lösungsvarianten wurden aufgezeigt. Abschließend wurden Möglichkeiten zur Erweiterung und Optimierung des heuristischen Verfahrens beschrieben.

Die Evaluierung enthält neben Details zur Implementierung der Konzepte und zur Durchführung der Experimente vor allem ausführliche Untersuchungen zur Funktionsweise und Performance der entwickelten Migrationsstrategien. Dabei wurden die Strategien zunächst im Einzelnen auf ihre korrekte Ausführung und ihre Laufzeiteigenschaften überprüft. Anschließend wurden die Ergebnisse der Leistungstests der neuen Strategien den Ergebnissen der existierenden Verfahren gegenübergestellt. Dabei bezogen sich die Tests auf periodische und aperiodische Datenströme wobei die Transferparameter sowohl unter Verwendung des numerischen als auch des heuristischen Verfahrens ermittelt wurden. Die dokumentierten Ergebnisse spiegeln hierbei eine repräsentative Teilmenge der im Prototyp und simulativ und mit verschiedenen Konfigurationen durchgeführten Experimente wider. Die Resultate zeigen, dass mit den neuen Strategien eine bessere Performance gegenüber den existierenden Migrationsverfahren erreicht wird bei gleichzeitiger Erfüllung aller Anforderungen.

Zusammenfassend kann festgehalten werden, dass mit dem neu entwickelten Migrationsverfahren eine Optimierung der Migration möglich ist und für zahlreiche, verschiedenartige Anwendungsbeispiele gezeigt wurde. Neben verbesserten Änderungsmöglichkeiten von Anfragen konnte auch die Migrationsdauer erheblich reduziert werden. Die Anpassungsmethodik erlaubt das Nachladen von Verarbeitungslogik zur Laufzeit und eine Durchführung der Migration bei gleichzeitiger Ergebnisausgabe. Mit Hilfe der Zustandstransferstrategien lässt sich die Übertragungsreihenfolge der Zustandswerte beeinflussen und die Transfermenge auf ein Minimum reduzieren. Für die Bestimmung der Transferparameter wurden zwei Verfahren vorgestellt, wodurch für alle Datenstromtypen eine

Möglichkeit zur Bestimmung einer geeigneten Migrationskonfiguration besteht. Dabei sind die Berechnungsverfahren auch für eine Anwendung auf verteilte Datenstromsysteme ausgelegt, da die Zeit, die für den Zustandstransfer benötigt wird, bei der Berechnung der Migrationsparameter berücksichtigt wird.

8.2 Ausblick

Der gegenwärtige Stand stellt eine geeignete Grundlage zur weiteren Optimierung der Migration in Datenstromsystemen dar. Aufgrund der Vielfältigkeit von Datenstromoperatoren, Datenstromtypen, Fenstereigenschaften usw. und den daraus resultierenden Kombinationsmöglichkeiten besteht ein Potenzial für Verbesserungen und Erweiterungen, von denen im Verlauf der Arbeit bereits einige beschrieben wurden. Nachfolgend werden die wichtigsten Punkte wiederholt und verschiedene Entwicklungsrichtungen skizziert.

Dynamische Anpassung der Migrationsparameter Die Berechnung der Migrationskonfiguration in aperiodischen Datenströmen basiert auf Näherungen, z.B. mittlerer Ereignisabstand oder durchschnittliche Pausendauer. Da sich diese Eigenschaften während des Zustandstransfers ändern, könnte ein dynamische Strategie darauf reagieren und mehr oder weniger Werte übertragen, d. h. die Transferparameter werden auch während des Zustandstransfers immer wieder neu abgeschätzt. Es ist zu untersuchen, unter welchen Bedingungen eine dynamische Strategie Vorteile bringt und inwiefern die Migrationsdauer verkürzt werden kann.

Systematische Analyse von Anfragen mit heterogenen Eigenschaften Für Anfragen mit heterogenen Eigenschaften, z. B. unterschiedliche Fenstergrößen, Selektivitäten oder Datenstrom-, Operator- und Fenstertypen, wurden einfache Migrationsszenarien unter Verwendung des heuristischen Verfahrens stichprobenartig untersucht. Die beobachteten Ergebnisse waren ähnlich positiv wie für die dokumentierten Fälle. Um eventuelle Probleme aufzudecken, Ausnahmefälle zu finden und allgemeingültige Aussagen über das Migrationsverhalten zu treffen, müssen solche Szenarien jedoch systematisch analysiert werden. Basierend auf den Erkenntnissen können dann Verbesserungen an den Verfahren zur Migrationsparameterbestimmung durchgeführt oder neue Zustandstransferstrategien entwickelt werden.

Auch im Zusammenhang mit dem Überschätzen von N_o und der daraus resultierenden verlängerten Migrationsdauer sind weiterführende Untersuchungen sinnvoll. Ebenso hat die Untersuchung der korrigierten Zustandstransferdauer bei Migrationsszenarien mit mehreren Zuständen gezeigt, dass die Migrationsdauer dadurch zwar reduziert werden kann, jedoch auch, dass Verbesserungspotenzial in Bezug auf die Migrationsdauer vorhanden ist. Bei Anfragen mit heterogenen Eigenschaften, könnte sich der Einfluss dieser Effekte vergrößern, weshalb eine verbesserte Methode dann unabdingbar wird.

Bei der Untersuchung weiterer Datenstromtypen können Szenarien mit Burst-Verhalten einbezogen werden. Da während einer Burst-Phase das System unter Volllast läuft, können aufgrund der geringeren Priorität des Zustandstransfers keine Zustandswerte übertragen werden. Für anzahlbasierte Fenster bedeutet dies, dass sie schneller als erwartet mit neuen Werten gefüllt werden. Das bedeutet aber auch, dass am Ende weniger Zustandswerte benötigt werden, um das Fenster zu vervollständigen. Mit der Strategie LF sollte die Migration problemfrei verlaufen, mit anderen Strategien dauert die Migration aber unter Umständen länger. Um flexibel auf derartige Änderungen reagieren zu können, wäre auch hier eine dynamisch anpassbare Strategie sinnvoll. Insofern wird eine Analyse von Migrationsvorgängen in Burst-Szenarien interessante Ergebnisse liefern, zumal ein Burst-Verhalten in realen Systemen nichts Ungewöhnliches ist.

Dynamischer Algorithmenwechsel Eine bekannte Methode zur Lastreduzierung bei erhöhter Systemlast ist das Wechseln zu schnelleren aber weniger genauen Algorithmen. Sobald die Systemlast wieder normal ist, wird zum Standardalgorithmus zurück gewechselt. Diese Funktionalität wird z. B. vom System OSIRIS unterstützt, jedoch mit der Einschränkung, dass die verwendeten Algorithmen bereits verfügbar sind. Ein Nachladen von Algorithmen zur Laufzeit ist nicht möglich. Die in dieser Arbeit implementierte Adaptionsmethodik kann zwar Algorithmen zur Laufzeit nachladen, verfügt zur Zeit jedoch nicht über eine Funktion zum dynamischen Algorithmenwechsel, da die Optimierung der Anfrageausführung nicht mit Mittelpunkt der Arbeit stand. Eine Kombination beider Methoden schafft neue Möglichkeiten bei der automatischen Anfrageoptimierung. Algorithmen lassen sich iterativ optimieren und bei Bedarf in den automatischen Lastausgleichsprozess integrieren.

Modifizieren von Werten beim Zustandstransfer Beim Zustandstransfer werden die ausgewählten Werte gegenwärtig einzeln und mit allen Attributen übertragen. Dies ist unter Umständen jedoch nicht notwendig. Wurde eine Anfrage so verändert, dass nicht alle Attribute der Tupel benötigt werden, kann durch Konvertieren der Tupel ihre Größe minimiert werden, wodurch sich auch ihre Transferzeit verringert. Sollte eine solche Funktion unterstützt werden, müssen gegebenenfalls die Verfahren zur Berechnung der Migrationsparameter angepasst werden.

Eine andere Möglichkeit zur Verkleinerung der Tupeltransferzeit ist die Verwendung eines Kompressionsverfahrens. Tupel werden dann vor dem Transfer vom Zustandssender komprimiert – einzeln oder in Gruppen – und beim Zustandsempfänger wieder dekomprimiert und in den Zustand eingefügt. Durch die erhöhte Transferkapazität kann die Migrationsdauer reduziert werden, zumindest bei Fensterverkleinerungen und für gleichgroße Fenster.

Besonders in aperiodischen Datenströmen können durch den Transfer von größeren komprimierten Tupelgruppen interessante Situationen entstehen, da je nach Kollisionsbehandlung Eingangswerte unter Umständen stark verzögert bearbeitet werden können

oder ganze Tupelgruppen neu übertragen werden müssen, da ihr Transfer zugunsten der regulären Verarbeitung abgebrochen wurde. Die durch die Komprimierung gewonnene Zeit könnte so schnell durch zusätzliche Transfers aufgebraucht sein. Hier gilt es, einen geeigneten Kompromiss zu finden.

Kombination von Verteilungsplanungsmethoden mit Migrationsverfahren Anfrageoptimierung und Lastverteilung beinhalten Methoden zum automatischen Verschieben von Operatoren auf andere Verarbeitungsknoten. Für die Verteilungsplanung werden dabei die Eigenschaften der Verarbeitungsknoten mit den Anforderungen der Operatoren verglichen, z. B. in Bezug auf Rechenleistung, Arbeitsspeicher oder Netzwerkeigenschaften. Von einer Kombination von Verteilungsplanungsmethoden mit Migrationsverfahren könnten beide profitieren. Die Eigenschaften der Systeme, die in der Verteilungsplanung modelliert sind, können vom Migrationsverfahren genutzt werden und ermöglichen gegebenenfalls eine bessere Abschätzung der Migrationsparameter. Umgekehrt helfen die Migrationsverfahren bei der Durchführung von Änderungen und außerdem bei der Vorhersage, wann Änderungen beendet sind und die Anfragen einen stationären Zustand erreicht haben. Diese Daten können wiederum in die Optimierung der Verteilungsplanung einfließen.

Änderungsverfolgung und Änderungsvisualisierung Änderungen an Operatoren können nachfolgende Operatoren beeinflussen. Vor allem in komplexen Anfragen ist dieser Einfluss schwierig auszumachen. Als Beispiel kann man sich eine einfache Anfrage zur Anlagenüberwachung vorstellen mit einem Operator, der den Mittelwert eines Sensorsignals mit Hilfe eines gleitenden Fensters ermittelt, und einem nachfolgenden Operator, der den Mittelwert mit einem Grenzwert vergleicht und bei Überschreitung eine Alarmnachricht an ein angeschlossenes System übermittelt. Das Signal sei tendenziell steigend. Vergrößert man nun im Zuge einer Anfrageoptimierung die Fenstergröße des ersten Operators, werden mehr alte Werte für die Berechnung verwendet. Bei einem tendenziell steigenden Signal bedeutet das, dass der Mittelwert etwas kleiner ist. Aus diesem Grund müsste der Grenzwert im nachfolgenden Operator gegebenenfalls auch reduziert werden, um ein Alarmsignal rechtzeitig zu senden. Die Entwicklung eines Verfahrens zur Identifikation solcher und ähnlicher Einflüsse sowie die Implementierung als Entscheidungsunterstützungssystem wären aus Nutzersicht sinnvoll.

Ein weiterer Aspekt bei Anfragenänderungen ist die Unterstützung des Benutzers beim Nachvollziehen und Bewerten von durchgeführten Änderungen. Dies erfordert zunächst, dass alle Änderungen aufgezeichnet werden, u. a. mit Informationen zu betroffenen Anfragen und Operatoren, zur Migration oder zu den Umständen der Änderung. So lassen sich anschließend Änderungsverläufe für Anfragen oder Operatoren visualisieren. Sofern Datenströme über einen erweiterten Zeitraum der Änderung gespeichert wurden, können diese mit den Änderungsereignissen gegenübergestellt werden. Dies kann dazu dienen, die Qualität der Ergebnisse vor und nach der Änderung zu vergleichen und Änderungen

zu bewerten. Somit können Nutzern Informationen bei der Optimierung von Anfragen zur Verfügung gestellt werden, z. B. um weitere Optimierungsschritte zu planen oder bei Bedarf Änderungen rückgängig zu machen. Das Rückgängigmachen von Änderungen erfordert unter Umständen , dass Ergebnisse oder auch innere Zustände korrigiert werden (reconciliation [BBMS08]). Insbesondere für die Änderung der inneren Zustände können Migrationsverfahren hilfreich sein.

Literaturverzeichnis

[AAB+05] Abadi, Daniel J. ; Ahmad, Yanif ; Balazinska, Magdalena ; Çetintemel, Uğur ; Cherniack, Mitch ; Hwang, Jeong-Hyon ; Lindner, Wolfgang ; Maskey, Anurag S. ; Rasin, Alexander ; Ryvkina, Esther ; Tatbul, Nesime ; Xing, Ying ; Zdonik, Stan: The Design of the Borealis Stream Processing Engine. In: *CIDR 2005, Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, `www.cidrdb.org`, January 4–7 2005, S. 277–289. – Online Proceedings

[ABB+04] Arasu, Arvind ; Babcock, Brian ; Babu, Shivnath ; Cieslewicz, John ; Datar, Mayur ; Ito, Keith ; Motwani, Rajeev ; Srivastava, Utkarsh ; Widom, Jennifer: STREAM: The Stanford Data Stream Management System / Stanford InfoLab. Version: 2004. `http://ilpubs.stanford.edu:8090/641/`. Stanford InfoLab, 2004 (2004-20). – Technical Report

[ACÇ+03] Abadi, Daniel J. ; Carney, Don ; Çetintemel, Uğur ; Cherniack, Mitch ; Convey, Christian ; Lee, Sangdon ; Stonebraker, Michael ; Tatbul, Nesime ; Zdonik, Stan: Aurora: a new model and architecture for data stream management. In: *The VLDB Journal* 12 (2003), August, Nr. 2, S. 120–139. – DOI 10.1007/s00778–003–0095–z. – ISSN 1066–8888 (Print) 0949–877X (Online)

[AH00] Avnur, Ron ; Hellerstein, Joseph M.: Eddies: Continuously Adaptive Query Processing. In: *SIGMOD Rec.* 29 (2000), June, Nr. 2, S. 261–272. – DOI 10.1145/335191.335420. – ISSN 0163–5808

[Aur09] *The Aurora Project*. `http://www.cs.brown.edu/research/aurora/`, June 2009

[AWHK07] Anke, Jürgen ; Wolf, Bernhard ; Hackenbroich, Gregor ; Kabitzsch, Klaus: A Planning Method for Component Placement in Smart Item Environments using Heuristic Search. In: Indulska, Jadwiga (Hrsg.) ; Raymond, Kerry (Hrsg.): *Distributed Applications and Interoperable Systems, 7th IFIP WG 6.1 International Conference, DAIS 2007, Paphos, Cyprus, June 6-8, 2007. Proceedings* Bd. 4531. Paphos, Cyprus : Springer, June 5-8 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–72881–8, S. 309–322

[BBC+04] Balakrishnan, Hari ; Balazinska, Magdalena ; Carney, Don ; Çetintemel, Uğur ; Cherniack, Mitch ; Convey, Christian ; Galvez, Eddie ; Salz, Jon ; Stonebraker, Michael ; Tatbul, Nesime ; Tibbetts, Richard ; Zdonik, Stan: Retrospective on Aurora. In: *The VLDB Journal* 13 (2004), Dezember, Nr. 4, S. 370–383. – DOI 10.1007/s00778–004–0133–5. – ISSN 1066–8888 (Print) 0949–877X (Online)

[BBD+02] Babcock, Brian ; Babu, Shivnath ; Datar, Mayur ; Motwani, Rajeev ; Widom, Jennifer: Models and Issues in Data Stream Systems. In: *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York, NY, USA : ACM, June 3–5 2002. – ISBN 1–58113–507–6, S. 1–16

[BBMS05] Balazinska, Magdalena ; Balakrishnan, Hari ; Madden, Samuel ; Stonebraker, Michael: Fault-Tolerance in the Borealis Distributed Stream Processing System. In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, June 13–16 2005. – ISBN 1–59593–060–4, S. 13–24

[BBMS08] Balazinska, Magdalena ; Balakrishnan, Hari ; Madden, Samuel R. ; Stonebraker, Michael: Fault-Tolerance in the Borealis Distributed Stream Processing System. In: *ACM TODS* 33 (2008), März, Nr. 1, S. 1–44. – DOI 10.1145/1331904.1331907. – ISSN 0362–5915

[BDD+10] Botan, Irina ; Derakhshan, Roozbeh ; Dindar, Nihal ; Haas, Laura ; Miller, Renée J. ; Tatbul, Nesime: SECRET: A Model for Analysis of the Execution Semantics of Stream Processing Systems. In: *Proceedings of the VLDB Endowment* 3 (2010), September, Nr. 1–2, S. 232–243. – ISSN 2150–8097

[Beh10] Behrens, Ingo: *Erweiterung und Implementierung von Konzepten zur Laufzeitadaption eines Datenstromsystems*, Otto-von-Guericke-Universität Magdeburg, Diplomarbeit, Februar 2010

[BFF09a] Brito, Andrey ; Fetzer, Christof ; Felber, Pascal: Minimizing Latency in Fault-Tolerant Distributed Stream Processing Systems. In: *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems*. Washington, DC, USA : IEEE Computer Society, Juni 22–26 2009. – ISBN 978–0–7695–3659–0, S. 173–182

[BFF09b] Brito, Andrey ; Fetzer, Christof ; Felber, Pascal: Multithreading-Enabled Active Replication for Event Stream Processing Operators. In: *Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems*. Washington, DC, USA : IEEE Computer Society, September 27–30 2009. – ISBN 978–0–7695–3826–6, S. 22–31

[BMM+04] Babu, Shivnath ; Motwani, Rajeev ; Munagala, Kamesh ; Nishizawa, Itaru ; Widom, Jennifer: Adaptive Ordering of Pipelined Stream Filters. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, Juni 13–18 2004. – ISBN 1–58113–859–8, S. 407–418

[BMWM05] Babu, Shivnath ; Munagala, Kamesh ; Widom, Jennifer ; Motwani, Rajeev: Adaptive Caching for Continuous Queries. In: *Proceedings of the 21st International Conference on Data Engineering, 2005. ICDE 2005.* Los Alamitos, CA, USA : IEEE Computer Society, April 5–8 2005. – ISBN 0–7695–2285–8, S. 118–129

[BO03] Babcock, Brian ; Olston, Chris: Distributed Top-K Monitoring. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, Juni 09–12 2003. – ISBN 1–58113–634–X, 28–39

[Bor09] *The Borealis Project*. `http://www.cs.brown.edu/research/borealis/`, June 2009

[Boy08a] Boyer, Brent: Robust Java benchmarking, Part 1: Issues / IBM developerWorks. Version: Juni 24 2008. `http://www.ibm.com/developerworks/java/library/j-benchmark1/`. 2008. – Whitepaper

[Boy08b] Boyer, Brent: Robust Java benchmarking, Part 2: Statistics and solutions / IBM developerWorks. Version: Juni 24 2008. `http://www.ibm.com/developerworks/java/library/j-benchmark2/`. 2008. – Forschungsbericht

[Boy08c] Boyer, Brent: *Robust Java benchmarking, Supplement*, 2008. `http://www.ellipticgroup.com/html/benchmarkingArticle.html`

[Bre08] Brettlecker, Gert: *Efficient and Reliable Data Stream Management*, Universität Basel, Departement Informatik, Dissertation, Mai 2008

[BSS06] Brettlecker, Gert ; Schek, Hans-Jörg ; Schuldt, Heiko: Eine Pervasive-Healthcare-Infrastruktur für die verlässliche Informationsverwaltung und -verarbeitung im Gesundheitswesen. In: *Datenbank-Spektrum* 6 (2006), Nr. 17, S. 33–41. – ISSN 1618–2162

[BSW04] Babu, Shivnath ; Srivastava, Utkarsh ; Widom, Jennifer: Exploiting k-Constraints to Reduce Memory Overhead in Continuous Queries Over Data Streams. In: *ACM TODS* 29 (2004), September, S. 545–580. – DOI 10.1145/1016028.1016032. – ISSN 0362–5915

[BW04] Babu, Shivnath ; Widom, Jennifer: StreaMon: An Adaptive Engine for Stream Query Processing. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, Juni 13–18 2004. – ISBN 1–58113–859–8, S. 931–932

[CAP09] *The CAPE Project*. `http://davis.wpi.edu/dsrg/PROJECTS/CAPE/index.html`, September 2009

[Car05] Carnero, M.C.: Selection of diagnostic techniques and instrumentation in a predictive maintenance program. A case study. In: *Decision Support Systems* 38 (2005), Nr. 4, S. 539–555. – DOI 10.1016/j.dss.2003.09.003. – ISSN 0167–9236

[CBB+03] Cherniack, Mitch ; Balakrishnan, Hari ; Balazinska, Magdalena ; Carney, Don ; Çetintemel, Uğur ; Xing, Ying ; Zdonik, Stan: Scalable Distributed Stream Processing. In: *CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research*, `www.cidrdb.org`, January 5–8 2003. – Online Proceedings

[CÇC+02] Carney, Don ; Çetintemel, Uğur ; Cherniack, Mitch ; Convey, Christian ; Lee, Sangdon ; Seidman, Greg ; Stonebraker, Michael ; Tatbul, Nesime ; Zdonik, Stan: Monitoring Streams: A New Class of Data Management Applications. In: *VLDB 2002: Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB Endowment, August 20–23 2002, S. 215–226

[CCD+03] Chandrasekaran, Sirish ; Cooper, Owen ; Deshpande, Amol ; Franklin, Michael J. ; Hellerstein, Joseph M. ; Hong, Wei ; Krishnamurthy, Sailesh ; Madden, Sam ; Raman, Vijayshankar ; Reiss, Fred ; Shah, Mehul: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: *Proceedings of the 2003 CIDR Conference*, `www.cidrdb.org`, January 5–8 2003. – Online Proceedings

[CEB+09] Cipriani, Nazario ; Eissele, Mike ; Brodt, Andreas ; Grossmann, Matthias ; Mitschang, Bernhard: NexusDS: A Flexible and Extensible Middleware for Distributed Stream Processing. In: *Proceedings of the 2009 International Database Engineering & Applications Symposium*. New York, NY, USA : ACM, September 16–18 2009 (IDEAS '09). – ISBN 978–1–60558–402–7, S. 152–161

[CJ09] Chakravarthy, Sharma ; Jiang, Qingchun: *Advances in Database Systems*. Bd. 36: *Stream Data Processing: A Quality of Service Perspective*. Springer US, 2009. – 324 S. DOI 10.1007/978–0–387–71003–7. – ISBN 978–0–387–71002–0 (Print) 978–0–387–71003–7 (Online)

[CLM10] Cipriani, Nazario ; Lübbe, Carlos ; Moosbrugger, Alexander: Exploiting Constraints to Build a Flexible and Extensible Data Stream Processing Middleware. In: *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and PhD Forum (IPDPSW)*, IEEE Computer Society, April 19–23 2010. – ISBN 978–1–4244–6533–0

[CNG+09] Cipriani, Nazario ; Nicklas, Daniela ; Grossmann, Matthias ; Hönle, Nicola ; Lübbe, Carlos ; Mitschang, Bernhard: Verteilte Datenstromverarbeitung von Sensordaten. In: *Datenbank-Spektrum* 9 (2009), Februar, Nr. 28, S. 37–43. – ISSN 1618–2162

[Cor12] Corrigan, Joseph P.: *Traffic Projections for 2012 & 2013*. Options Price Reporting Authority. `http://www.opradata.com/specs/2012_2013_Traffic_Projections.pdf`. Version: Januar 2012

[DIR07] Deshpande, Amol ; Ives, Zachary ; Raman, Vijayshankar: *Adaptive Query Processing*. Hanover, MA, USA : Now Publishers Inc., 2007. – 140 S. DOI 10.1561/1900000001. – ISBN 978–1–60198–034–2

[Eqnx11] The Eclipse Foundation: *Eclipse Equinox*. `http://www.eclipse.org/equinox`, Juni 2011

[Fri02] Friedman, Alan: What Is The "Crest Factor" And Why Is It Used? / AZIMA DLI. Version: 2002. `http://www.dliengineering.com/published/TECH-CrestFactor.pdf`. 2002. – Technical Report

[GB12] Geisberger, Eva ; Broy, Manfred: agendaCPS – Integrierte Forschungsagenda Cyber-Physical Systems / acatech – Deutsche Akademie der Technikwissenschaften. 2012. – Report

[GG07] Gama, João (Hrsg.) ; Gaber, Mohamed M. (Hrsg.): *Learning from Data Streams: Processing Techniques in Sensor Networks*. Springer Berlin Heidelberg, 2007. – 244 S. DOI 10.1007/3–540–73679–4. – ISBN 978–3–540–73678–3 (Print) 978–3–540–73679–0 (Online)

[GGM+04] Green, Todd J. ; Gupta, Ashish ; Miklau, Gerome ; Onizuka, Makoto ; Suciu, Dan: Processing XML Streams With Deterministic Automata and Stream Indexes. In: *ACM TODS* 29 (2004), Dezember, Nr. 4, S. 752–788. – DOI 10.1145/1042046.1042051. – ISSN 0362–5915

[GoF94] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John M.: *Design Patterns: Elements of Reusable Object-Oriented Software*. First. Addison-Wesley Professional, 1994. – 416 S. – ISBN 0–201–63361–2

[GÖ03a] Golab, Lukasz ; Özsu, M. T.: Issues in Data Stream Management. In: *SIGMOD Rec.* 32 (2003), June, Nr. 2, S. 5–14. – DOI 10.1145/776985.776986. – ISSN 0163–5808

[GÖ03b] Golab, Lukasz ; Özsu, M. T.: Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. In: *VLDB 2003: Proceedings of the 29th International Conference on Very Large Data Bases* Bd. 29, VLDB Endowment, September 9–12 2003. – ISBN 0–12–722442–4, S. 500–511

[Ham97] Hamann, Claude-Joachim: On the Quantitative Specification of Jitter Constrained Periodic Streams. In: *Proceedings of the Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'97)*, IEEE Computer Society, Januar 12–15 1997. – ISBN 0–8186–7758–9, S. 171–176

[Ham05] Hamann, Claude-Joachim: Schwankungsbeschränkte Datenströme / Technische Universität Dresden, Fakultät Informatik, Institut für Systemarchitektur. 2005 (TUD-FI05-11). – Technischer Bericht. – ISSN 1430–211X

[HGA+07] Hammad, M. A. ; Ghanem, T. M. ; Aref, W. G. ; Elmagarmid, A. K. ; Mokbel, M. F.: Data Stream Management Systems and Architectures. Version: 2007. In: [GG07], Kapitel 5, S. 51–71

[HK05] Han, Jiawei ; Kamber, Micheline: *Data Mining: Concepts and Techniques (Second Edition)*. Second. Morgan Kaufmann, 2005 (The Morgan Kaufmann Series in Data Management Systems). – 800 S. – ISBN 978–1–55860–901–3

[Hol06] Holmes, David: *Inside the Hotspot VM: Clocks, Timers and Scheduling Events*. Weblog. `http://blogs.sun.com/dholmes/entry/inside_the_hotspot_vm_clocks`. Version: Oktober 02 2006

[IS00a] Isert, Carsten ; Schwan, Karsten: ACDS: Adapting Computational Data Streams for High Performance / Georgia Institute of Technology. Version: 2000. `http://hdl.handle.net/1853/6579`. 2000 (GIT-CC-00-01). – Technical Report

[IS00b] Isert, Carsten ; Schwan, Karsten: ACDS: Adapting Computational Data Streams for High Performance. In: *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*. Los Alamitos, CA, USA : IEEE Computer Society, Mai 1–5 2000. – ISBN 0–7695–0574–0, S. 641–646

[JLB06] Jardine, Andrew K. ; Lin, Daming ; Banjevic, Dragan: A review on machinery diagnostics and prognostics implementing condition-based maintenance. In: *Mechanical Systems and Signal Processing* 20 (2006), Nr. 7, S. 1483–1510. – DOI 10.1016/j.ymssp.2005.09.012. – ISSN 0888–3270

[KH04] Kriens, Peter ; Hargrave, B. J.: Listeners Considered Harmful: The "Whiteboard" Pattern / The OSGi Alliance. 2004. – Technical Whitepaper

[Krä07] Krämer, Jürgen: *Continuous Queries over Data Streams – Semantics and Implementation*, Philipps-Universität Marburg, Dissertation, 2007

[KS03] Kazzaz, Sa'ad A. S. A. ; Singh, G.K.: Experimental investigations on induction machine condition monitoring and fault diagnosis using digital signal processing techniques. In: *Electric Power Systems Research* 65 (2003), Nr. 3, S. 197–221. – DOI 10.1016/S0378–7796(02)00227–4. – ISSN 0378–7796

[KS04] Krämer, Jürgen ; Seeger, Bernhard: A Temporal Foundation for Continuous Queries over Data Streams / Department of Mathematics and Computer Science, University of Marburg. 2004 (No. 45). – Technical Report

[Mis06] Miskowicz, Marek: Send-On-Delta Concept: An Event-Based Data Reporting Strategy. In: *Sensors* 6 (2006), Januar, Nr. 1, 49–63. – DOI 10.3390/s6010049. – ISSN 1424–8220

[Mut05] Muthukrishnan, S.: *Data Streams: Algorithms and Applications*. now Publishers Inc., 2005. – 126 S. DOI 10.1561/0400000002

[MWA+03] Motwani, Rajeev ; Widom, Jennifer ; Arasu, Arvind ; Babcock, Brian ; Babu, Shivnath ; Datar, Mayur ; Manku, Gurmeet ; Olston, Chris ; Rosenstein, Justin ; Varma, Rohit: Query Processing, Resource Management, and Approximation in a Data Stream Management System. In: *CIDR 2003, Proceedings of the First Biennial Conference on Innovative Data Systems Research*, `www.cidrdb.org`, January 5–8 2003. – Online Proceedings

[Nel90] Nelson, Victor P.: Fault-Tolerant Computing: Fundamental Concepts. In: *Computer* 23 (1990), Juli, Nr. 7, S. 19–25. – DOI 10.1109/2.56849. – ISSN 0018–9162

[Neu06] Neugebauer, Mario: *Energieeffiziente Anpassung des Arbeitszyklus in drahtlosen Sensornetzen*, Technischen Universität Dresden, Fakultät Informatik, Dissertation, September 2006

[Noo09] Noortwijk, J.M. van: A survey of the application of gamma processes in maintenance. In: *RESS* 94 (2009), Nr. 1, S. 2–21. – DOI 10.1016/j.ress.2007.03.019. – ISSN 0951–8320

[OJW03] Olston, Chris ; Jiang, Jing ; Widom, Jennifer: Adaptive Filters for Continuous Queries over Distributed Data Streams. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, Juni 09–12 2003. – ISBN 1–58113–634–X, 563–574

[OSGi09] The OSGi Alliance: *OSGi Service Platform Core Specification*. Juni 2009. – Release 4, Version 4.2

[OSGi11] The OSGi Alliance: *OSGi*. `http://www.osgi.org`, Juni 2011

[QLLY03] Qiu, Hai ; Lee, Jay ; Lin, Jing ; Yu, Gang: Robust performance degradation assessment methods for enhanced rolling element bearing prognostics. In: *Advanced Engineering Informatics* 17 (2003), Nr. 3-4, S. 127–140. – DOI 10.1016/j.aei.2004.08.001. – ISSN 1474–0346

[RA10] Rodrigues Américo, Joao Claudio: *A study of the impact of real-time constraints in Java/OSGi applications*, Université Joseph Fourier, Grenoble, Masters Thesis, Juni 25 2010

[RDH03] Raman, Vijayshankar ; Deshpande, Amol ; Hellerstein, Joseph M.: Using State Modules for Adaptive Query Processing. In: *Proceedings of the 19th International Conference on Data Engineering*, IEEE Computer Society, März 5–8 2003. – ISBN 0–7803–7665–X, S. 353–364

[RDZ+05] Rundensteiner, Elke A. ; Ding, Luping ; Zhu, Yali ; Sutherland, Timothy ; Pielech, Bradford: CAPE: A Constraint-Aware Adaptive Stream Processing Engine. Version: 2005. In: Chaudhry, Nauman A. (Hrsg.) ; Shaw, Kevin (Hrsg.) ; Abdelguerfi, Mahdi (Hrsg.): *Stream Data Management* Bd. 30. Springer US, 2005. – DOI 10.1007/0–387–25229–0_5. – ISBN 978–0–387–24393–1 (Print) 978–0–387–25229–2 (Online), Kapitel 5, S. 83–111

[RF10] Ryll, Frank ; Freund, Curt: Grundlagen der Instandhaltung. Version: 2010. In: [Sch10], Kapitel 2, S. 23–101

[Ros08] Rosjat, Martin: *Prototypische Lösung eines dynamisch modifizierbaren, datenstromorientierten Systems auf der Basis des OSGi-Frameworks*, Westsächsische Hochschule Zwickau, Diplomarbeit, Juli 2008

[Sch07] Schrüfer, Elmar: *Elektrische Messtechnik: Messung elektrischer und nichtelektrischer Größen*. 9., aktualisierte Aufl. München : Hanser Fachbuchverlag, 2007. – ISBN 978–3–446–40904–0

[Sch10] Schenk, Michael (Hrsg.): *Instandhaltung technischer Systeme – Methoden und Werkzeuge zur Gewährleistung eines sicheren und wirtschaftlichen Anlagenbetriebs*. Berlin Heidelberg : Springer, 2010 DOI 10.1007/978–3–642–03949–2. – ISBN 978–3–642–03949–2

[SÇZ05] Stonebraker, Michael ; Çetintemel, Uğur ; Zdonik, Stan: The 8 Requirements of Real-time Stream Processing. In: *SIGMOD Rec.* 34 (2005), Dezember, Nr. 4, S. 42–47. – DOI 10.1145/1107499.1107504. – ISSN 0163–5808

[SE05] Shvaiko, Pavel ; Euzenat, Jérôme: A Survey of Schema-Based Matching Approaches. Version: 2005. In: Spaccapietra, Stefano (Hrsg.): *Journal on Data Semantics IV* Bd. 3730. Berlin / Heidelberg : Springer, 2005. – DOI 10.1007/11603412_5, S. 146–171

[SHB04] Shah, Mehul A. ; Hellerstein, Joseph M. ; Brewer, Eric: Highly Available, Fault-Tolerant, Parallel Dataflows. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, Juni 13–18 2004. – ISBN 1–58113–859–8, S. 827–838

[SHCF02] Shah, Mehul A. ; Hellerstein, Joseph M. ; Chandrasekaran, Sirish ; Franklin, Michael J.: Flux: An Adaptive Partitioning Operator for Continuous Query Systems / EECS Department, University of California, Berkeley. Version: Oktober 2002. `http://www.eecs.berkeley.edu/Pubs/TechRpts/2002/5689.html`. 2002 (UCB/CSD-02-1205). – Technical Report

[Spi01] Spitzer, Frank: *Graduate Texts in Mathematics*. Bd. 34: *Principles of Random Walk*. Second. Springer Verlag, 2001. – 408 S. – ISBN 978–0387951546

[SR04] Sutherland, Tim ; Rundensteiner, Elke A.: D-CAPE: A Self-Tuning Continuous Query Plan Distribution Architecture / Worcester Polytechnic Institute. 2004 (WPI-CS-TR-04-18). – Technical Report

[STR09] *STREAM – Stanford Stream Data Manager*. `http://infolab.stanford.edu/stream/`, Januar 2009

[Tel09] *TelegraphCQ*. `http://telegraph.cs.berkeley.edu/telegraphcq/v2.1/`, März 2009

[VNB03] Viglas, Stratis D. ; Naughton, Jeffrey F. ; Burger, Josef: Maximizing the Output Rate of Multi-way Join Queries over Streaming Information Sources. In: *VLDB 2003: Proceedings of the 29th International Conference on Very Large Data Bases* Bd. 29, VLDB Endowment, September 9–12 2003. – ISBN 0–12–722442–4, S. 285–296

[WB10] Wolf, Bernhard ; Behrens, Ingo: On-the-fly Adaptation of Data Stream Queries. In: *Proceedings of the 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2010)*. Carmona, Spain : IEEE Computer Society, Mai 5–6 2010. – ISSN 1555–0885, S. 175–179

[WDD94] Williams, John H. (Hrsg.) ; Davies, Alan (Hrsg.) ; Drake, Paul R. (Hrsg.): *Condition-based Maintenance and Machine Diagnostics*. London, UK : Chapman & Hall, 1994. – ISBN 978–0–412–46500–0

[WHB+10] Wolf, Bernhard ; Herzig, Philipp ; Behrens, Ingo ; Majumdar, Anirban ; Ameling, Michael: Data Stream Processing in Factory Automation. In: *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2010)*, IEEE, September 13–16 2010. – ISBN 978–1–4244–6848–5

[Wie09] Wiese, Franziska: *Entwicklung einer Kommunikationsschnittstelle zwischen OSGi und Adobe Flex zur Verwendung in einem dynamischen Datenstromsystem*, Technische Universität Dresden, Studienarbeit, April 2009

[WRBK01] Williams, T. ; Ribadeneira, X. ; Billington, S. ; Kurfess, T.: Rolling Element Bearing Diagnostics in Run-to-failure Lifetime Testing. In: *Mechanical Systems and Signal Processing* 15 (2001), Nr. 5, S. 979–993. – DOI 10.1006/mssp.2001.1418. – ISSN 0888–3270

[WVV+01] Wache, H. ; Vögele, T. ; Visser, U. ; Stuckenschmidt, H. ; Schuster, G. ; Neumann, H. ; Hübner, S.: Ontology-Based Integration of Information – A Survey of Existing Approaches. In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 01)*, 2001, S. 108–117. – Workshop: Ontologies and Information Sharing

[YKPS07] Yang, Yin ; Krämer, Jürgen ; Papadias, Dimitris ; Seeger, Bernhard: HybMig: A Hybrid Approach to Dynamic Plan Migration for Continuous Queries. In: *IEEE Transactions on Knowledge and Data Engineering* 19 (2007), März, Nr. 3, S. 398–411. – DOI 10.1109/TKDE.2007.43. – ISSN 1041–4347

[Zhu06] Zhu, Yali: *Dynamic Optimization and Migration of Continuous Queries Over Data Streams*, Worcester Polytechnic Institute, Dissertation, August 2006

[ZRH04] Zhu, Yali ; Rundensteiner, Elke A. ; Heineman, George T.: Dynamic Plan Migration for Continuous Queries Over Data Streams. In: *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, Juni 13–18 2004. – ISBN 1–58113–859–8, S. 431–442